在线服务：视频库、源代码库、专业论坛、专家实时支持

Flex+ASP.NET Web应用开发实战详解



许勇　王黎　等编著



55段全程配音语音教学视频
全书实例源代码，使学习、分析、调试程序更方便

在线服务方式

在线服务网站：www.itzcn.com
QQ群在线服务：45368980、33925615、107423140

清华大学出版社

Flex+ASP.NET Web 应用开发实战详解

许　勇　王　黎　等编著

清华大学出版社
北　京

内 容 简 介

Flex 是开发富互联网应用程序（Rich Internet Application，RIA）的利器，它提供了丰富的可扩展用户界面及数据访问组件。本书共分为 5 篇，分别是 Flex 基础知识篇、ASP.NET 编程篇、Flex 组件应用篇、Flex 数据交互篇和综合实例篇。全书对 Flex 和 ASP.NET 两种流行技术进行了归纳和总结，内容覆盖了 Flex 和 ASP.NET 技术的知识和应用场景，力求通过实例使读者更形象地理解 ActionScript 的编程思想，快速掌握 Flex 的组件开发。

本书可作为 Flex 开发人员的重要学习资料，也可作为网站开发和 Flex 开发人员的职业培训教程。

图书在版编目（CIP）数据

Flex+ASP.NET Web 应用开发实战详解 / 许勇，王黎等编著. —北京：清华大学出版社，2010.7
ISBN 978-7-302-22316-0

Ⅰ. ①F…　Ⅱ. ①许…　②王…　Ⅲ. ①软件工具－程序设计②主页制作－程序设计　Ⅳ. ①TP311.56②TP393.092

中国版本图书馆 CIP 数据核字（2010）第 055946 号

责任编辑：夏兆彦
责任校对：徐俊伟
责任印制：
出版发行：清华大学出版社　　**地　　址：**北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：190×260　　**印　张：**34.75　　**插　页：**　　**字　数：**862 千字
版　　次：2010 年 7 月第 1 版　　**印　次：**2010 年 7 月第 1 次印刷
印　　数：
定　　价：　　元

产品编号：031595-01

FOREWORD

前言

Adobe Flex 是为满足希望开发富互联网应用程序的企业级程序员的需求而推出的表示服务器和应用程序框架。Flex 是开发富互联网应用程序（Rich Internet Application，RIA）的利器，它无缝整合了 Flash、ActionScript 以及 MXML，并提供了丰富的可扩展用户界面及数据访问组件，使开发人员能够快速构建出具有丰富数据演示、强大客户端逻辑和集成多媒体的 RIA 应用程序。目前已广泛应用于各种商业领域，如电子商务、行政管理、企业业务流程自动化等方面。

1．本书内容

本书对 Flex 和 ASP.NET 两种流行技术进行了归纳和总结，内容覆盖了 Flex 和 ASP.NET 技术的知识和应用场景。本书共分为 5 篇，分别是：Flex 基础知识篇、ASP.NET 编程篇、Flex 组件应用篇、Flex 数据交互篇和综合实例篇。各篇主要内容如下所示。

第 1 篇：Flex 基础知识篇（第 1～5 章）。主要介绍 Flex 中脚本语言 ActionScript 的知识，像 ActionScript 的运算符、控制语句、面向对象编程和内置类等。其中，第 1、2 章向读者介绍 Flex 环境的配置以及 Flex Builder 3.0 的基本操作。第 3、4 章主要介绍 ActionScript 的语法和面向对象的支持，像常量、变量、数据类型、运算符、对象、类、属性和方法等。第 5 章介绍如何使用函数、处理字符串、处理数组以及处理日期和时间。

第 2 篇：ASP.NET 编程篇（第 6～8 章）。主要针对 ASP.NET 初学者介绍开发常规网站所必备的基础知识，包括 ASP.NET 提供的环境的配置，C# 3.5 语法，ADO.NET 提供的对数据的查询、增加、删除、修改操作以及数据显示控件，XML 数据的解析、显示和生成，ASP.NET 的内置对象、Web 服务和文件操作。

第 3 篇：Flex 组件应用篇（第 9～13 章）。第 9、10 章详细介绍 Flex 中的基础类组件、布局类组件、导航和容器类组件。第 11 章主要介绍如何使用行为、对象状态和动画效果制作出绚丽的界面效果。第 12 章介绍 Flex 的事件机制，重点是使读者理解事件的工作流程、了解 Event 对象并掌握如何使用自定义事件。第 13 章介绍如何创建用户自定义组件，为组件定义 CSS 样式和主题，以及在组件之间进行参数传递。

第 4 篇：Flex 数据交互篇（第 14～15 章）。主要针对数据存储和交互这两个方面进行介绍，其中第 14 章向读者介绍在 Flex 3.0 中如何对数据进行处理，包括数据模型、数据绑定、DataGrid 组件、数据验证和数据格式化等内容。第 15 章介绍在 Flex 中如何与服务器端进行数据交互，包括常用的数据传输方式、使用 HTTPService 和 WebService 组件进行交互，并制作了留言本和域名查询实例。

第 5 篇：综合实例篇（第 16～17 章）。这一篇包括两个实例，第 16 章使用 Flex 制作一个功能齐全的 FLV 播放器，实现色彩调整、视频列表、播放控制等功能。第 17 章则是一个视频展示网站，在服务器端使用 ASP.NET 和 SQL Server，前台实现了用户注册和登录，视频分类、搜索、收藏、播放及视频列表等功能；后台实现了类别、视频的添加和管理操作。

2. 本书特色

书中采用大量的实例进行讲解，力求通过实例使读者更形象地理解 ActionScript 的编程思想，快速掌握 Flex 的组件开发方法。本书难度适中，内容由浅入深，实用性强，覆盖面广，条理清晰。

- **知识点全** 本书紧紧围绕利用 Flex 与 ASP.NET 进行 RIA 程序开发展开讲解，具有很强的逻辑性和系统性。
- **实例丰富** 书中各实例均经过作者精心设计和挑选，它们都是由作者在实际开发中的经验总结而来，涵盖了在实际开发中所遇到的各种问题。
- **应用广泛** 对于精选案例，给出了详细步骤，结构清晰简明、分析深入浅出，而且有些程序能够直接在项目中使用，避免读者进行二次开发。
- **基于理论，注重实践** 在讲述过程，不仅仅介绍理论知识，而且在合适位置安排综合应用实例，或者小型应用程序，将理论应用到实践当中，来加强读者实际应用能力，巩固 Flex 开发基础知识。
- **随书光盘** 本书为实例配备了视频教学文件，读者可以通过视频文件更加直观地学习 Flex 和 ASP.NET 的使用方法。
- **网站技术支持** 读者在学习或者工作的过程中，如果遇到实际问题，可以直接登录 www.itzcn.com 与我们取得联系，作者会在第一时间内给予帮助。

3. 读者对象

本书具有知识全面、实例精彩、指导性强的特点，力求以全面的知识及丰富的实例来指导读者透彻地学习 Flex 各方面的知识。本书可以作为 Flex 开发人员的重要学习资料，也可以作为网站开发和 Flex 开发人员的职业培训教程。

本书适合以下人员阅读学习。

- Flash 开发人员。
- Flex 应用开发人员。
- 网站建设及网络开发人员。
- Flex 开源项目爱好者。
- RIA 应用爱好者。

除了封面署名人员之外，参与本书编写的还有杨辉、胡家宏、于永军、张秋香、李乃文、张仕禹、夏小军、赵振江、李振山、李文才、吴越胜、李海庆、何永国、李海峰、陶丽、吴俊海、安征、张巍屹、崔群法、王咏梅、康显丽、辛爱军、牛小平、贾栓稳、王立新、苏静、赵元庆、郭磊、徐铭、李大庆、王蕾、张勇、郝安林、郭新志、牛丽平、唐守国等。在编写过程中难免会有疏漏之处，欢迎读者与我们联系，帮助我们改正提高。

CONTENTS

目录

第 1 篇　Flex 基础知识篇

第 4 篇 Flex 数据交互篇

第 5 篇 综合实例篇

第 1 篇　Flex 基础知识篇

第 1 章 Flex 3.0 入门



Flex 是一个提供开发设计和运行支持的架构，可以使开发人员创建利用 Adobe Flash Player 9 作为前台的 RIA（Rich Internet Application，富客户端互联网应用程序），以满足用户更为直观和极具交互性的在线体验的要求。

本章以最新的 Flex 3.0 为例向读者详细讲解什么是 Flex、Flex 的组成部分、如何配置 Flex 开发环境以及开发第一个实例程序。



了解 RIA 和 Flex 的概念
理解 Flex 架构及其各部分的作用
了解 Flex 3.0 的新特性
熟悉 MXML 的命名规范和文件结构
熟悉 ActionScript 的使用方式
掌握 Flex 3.0 的环境配置方法
开发第一个 Flex 程序

1.1 Flex 概述

在 Ajax 和 Microsoft WPF 出现之前，Macromedia（现为 Adobe）就曾推出基于 Flash 的 RIA 解决方案，用于创建具有富交互和多功能的 Web 应用程序（又称为 RIA）。现在，Adobe 对 Flash 进行了增强，使其具有超越 Web 的功能，从而成为一个完整的开发环境，这就是 Flex。

1.1.1 RIA 发展

用户与目前的 Web 应用程序交互时，其体验并不能令人满意，主要体现在，Web 模型是基于页面的模型，而且缺少客户端智能。因此即使是相对简单的事务处理（例如网上购物）也让人混淆。而且，它几乎无法完成复杂的用户交互（如传统的客户端/服务器应用程序和桌面应用程序中的用户交互）。这样的技术使得 Web 应用程序难以使用、支持成本高，并且在很多方面无法发挥作用。

为了提高用户体验，出现了一种新类型的 Internet 应用程序，那就是 Rich Internet Application（以下简称 RIA）。这些应用程序结合了桌面应用程序反应快、交互性强的优点与 Web 应用程序传播范围广及容易传播的特性。RIA 简化并改进了 Web 应用程序的用户交互，这样，用户开发的应用程序可以提供更丰富、更具有交互性和响应性的用户体验。

Macromedia 是公认的新兴 RIA 市场的领导者。今天 98%的浏览器上都使用 Macromedia Flash 客户端软件。因此几乎每个人都可以使用基于 Flash 的 RIA。Macromedia Flex 是 Macromedia 的新服务器产品，它使企业应用程序开发人员能够全面访问 RIA 的功能。Flex 具有基于标准的架构，与当前企业开发人员的工具、方法和设计模式互补。

1. 过渡

从最初的 HTML 到现在，服务器端系统架构经历了很多次重要转变。在此过程中，客户端的表现功能也有一些转变，并且每个阶段的计算功能所带来的应用程序体验也有变化，直到 RIA 出现为止。下面列举其中重要的 3 个方面。

基于主机的应用程序

由基于主机的计算发展而来的交互式应用程序。推动此阶段计算的商业需求来自于企业组织内部业务自动化，例如工资表。应用程序在内部的专用网络间进行本地发布，用户界面的丰富性仅限于文本范围内。

客户端/服务器应用程序

这种模式发展得很快，主要是因为需要对企业组织内部的信息及应用进行部门级别的访问。仍然是在企业组织内部本地访问应用程序，但随着图形用户界面的出现及客户端处理的应用，应用程序的丰富性大大提高。

Web 应用

Web 应用程序的全局性应用，以及基于主机集中管理应用程序的模式，突破了客户端/服务器模式的主要限制。但对于用户来说，这需要很大的投入。从处理的角度来看，Web 应用程序模式将客户端转变为虚拟终端。用于提供最佳用户体验的主要交互问题消失了，这些主要问题包括直接控制、客户端处理及局部存储等。

2. RIA 出现

Macromedia 公司于 2001 年初首先提出了 RIA 的概念。当时走在前沿的 Flash 开发者们都已经开始在实际应用开发中使用类似的模型来架构他们的程序。这些应用与传统的基于 HTML 的 Web 应用相比，扩展了设计的自由度，突破了用户的交互局限。

虽然 RIA 的优势很明显，但要求新技术要与现有的基础结构及处理过程相适合。这就需要满足如下要求。

提供一个用户熟悉的编程模型

基于熟悉的面向对象的语言（像 Java 或者 C#）进行业务逻辑的后台开发，及利用基于标记的语言（像 XHTML）进行用户界面开发。新产品必须以现有的模型为基础进行创建，以充分利用现有的技术并确保较低的成本。

利用现有的基础结构

很多企业组织在服务器应用技术上进行了大量投资，而且大部分企业组织将在内部使用

J2EE 及.NET 技术开发产品。多数的企业组织也需要使用并符合该基础结构。

采用标准的协议及 API

Web 的众多优点之一就是在整个技术中采用更宽的标准。这些标准包括但不局限于行业标准，如 HTML/HTTP、XML、SOAP、Web Service、CSS，还有 J2EE 及.NET 等。

保留现有的工具

对于采用表示层解决方案的开发人员来说，关键问题是能否确保使用现有的编辑器或集成开发环境（IDE）来编写应用程序。开发人员希望利用主要 IDE（如 Eclipse、Visual Studio）和主要的 Web 应用程序开发产品（如 Dreamweaver）来编写代码。

高效的生产工具

生产工具帮助人们使用新技术进行高效率的工作。它们还帮助人们学习新技术中新的语言、框架和结构，并指导人们进行最佳操作。

支持主要的设计模型

企业在创建应用程序时使用设计模型进行企业开发。在过去的几年中，类似模型——视图-控制器（MVC）的模型在 J2EE 和.NET 企业应用程序开发中越来越普遍。表示层解决方案必须适应现有的这些基于模型的架构。

由于架构一致性与基于标准的解决方案的要求，RIA 一直无法为多数的企业应用程序开发人员所利用。这种情况直到最近才有所改变。

1.1.2　Flex 简介

Flex 是 Adobe 的新产品，它满足了某些 IT 开发人员的需要。他们希望开发一种应用程序，这种程序既有桌面应用程序的响应性与丰富性，又具有 Web 传播范围广的特性，即 RIA。

Flex 驻留在组织的 N 层应用程序模型的表示层，使用在客户端运行的可执行代码，作为当前 HTML 的补充。Flex 具有基于标准的、用户熟悉的编程方法及工作流，强大的类库可创建表示层，从而提供更有效、更真实的终端用户体验。

Flex 应用程序与传统的 HTML 应用程序的主要区别在于 Flex 应用程序处理最适合在客户端运行，如字段校验、数据格式、分类、过滤、工具提示、合成视频、行为及效果等。Flex 可使开发人员更好地执行应用程序，这种应用程序使用户可以迅速反应、在不同状态与显示时流畅过渡，并提供毫无中断的连续性工作流。

1. Flex 开发模型

Flex 开发模型对于使用 JSP、ASP/ASP.NET 或其他类似的脚本语言的开发人员来说并不陌生。基本的模型相同：建立一个包含应用程序源代码的文本文件，然后将此文件部署到服务器上；服务器在收到第一个请求时，将此源码编译成为应用程序，后续的请求将经过缓存处理。与发送一系列包含数据与布局的 HTML 页面不同，Flex 发送可在 Flash Player 虚拟机上运行的富客户端用户界面。需要时，Flex 应用程序将与服务器交换数据，以响应客户端上终端用户的操作。

Flex 开发者使用扩展的用户界面组件库与基于 XML 标记的语言定义用户界面，利用面向对象的脚本语言（ActionScript）来处理程序逻辑。

除了在现有的表示层上进行添加外，Flex 并不需要对当前的业务层与整合层进行任何改变。Flex 在应用服务器内运行，并为 Flex 应用程序提供整合与管理能力。Flex 整合的能力可以轻松地通过 Web 服务、Java 对象访问或 XML 使用现有的代码及信息。Flex 还可以与一些现有的表示层技术与框架结构（如 JSP 及 Struts 等）进行集成。

2. Flex 的部署和管理

在 J2EE 平台上部署 Flex 服务器非常简单，因为 Flex 是本地 Java 应用程序。在 J2EE 平台上部署 Flex 应用程序是通过 Java Web 包（WAR）文件处理的。从管理角度看，Flex XML 方案和基于文件的应用程序模型意味着，Flex 应用程序可以与现有的管理工具和应用程序生命周期工具轻松集成。在 Flash Player 中执行时，Flex 应用程序可以与服务器端的功能交互，这些功能包括 Java 对象、SOAP Web 服务和其他服务器端服务。

1.1.3 Flex 架构

在图 1-1 中显示了 Flex 的基本架构，可以看到 Flex 主要由 Flex 应用程序框架与 Flex 运行时服务构成。下面将针对架构中的重要部分进行简单介绍，首先是 Flex 应用程序框架。



图 1-1 Flex 基本架构

1. Flex 应用程序框架

如图 1-1 所示，Flex 应用程序框架由 MXML、ActionScript 及 Flex 类库构成。开发人员利用 MXML 及 ActionScript 编写 Flex 应用程序。利用 MXML 定义应用程序用户界面元素，利用 ActionScript 定义客户逻辑与程序控制。Flex 类库中包括 Flex 组件、管理器及行为等。利用基于 Flex 组件的开发模型，开发人员可在程序中加入内置的组件、创建新组件或是将内置的组件加入复合组件中。

2. MXML：Flex 标记语言

MXML 和 HTML 一样，是标记语言，它描述了反映内容与功能的用户界面。与 HTML

不同的是，MXML 为表示层用户界面和服务器端数据绑定提供了声明的抽象。MXML 可将表示与业务逻辑的问题彻底分开，以实现最大程度的提高开发人员的生产率及应用程序的重复使用率。

MXML 的开发基础是在迭代过程上。这与其他类型的 Web 应用程序文件相同，如 HTML、JSP、ASP 及 ColdFusion 标记语言（CFML）。开发 MXML 应用程序就像打开一个文本编辑器一样简单，只要输入一些标签，保存文件，然后在 Web 浏览器上打开文件 URL 即可。

另外，MXML 文件同时也是普通的 XML 文件，所以可以选择多种开发环境。可以在简单文件编辑器、专用 XML 编辑器或是支持文件编辑的集成开发环境（例如 Flex Builder）中进行开发。由于 MXML 符合 W3C XML 方案的定义，也可以使用结构化编辑，如代码着色和代码提示（取决于编辑器的功能）。图 1-2 所示为片段 MXML 在开发环境 Flex Builder 中的效果。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"  creationComplete='
<mx:HTTPService showBusyCursor="true"   id="getUser"
    url="http://localhost/flex/test/Default.aspx" result="CheckUserProc()">
    <mx:request xmlns="">
        <username>
            {this.txtUserName.text}
        </username>
        <userpwd>
            {this.txtUserPwd.text}
        </userpwd>
    </mx:request>
</mx:HTTPService>
<mx:HTTPService id="dsClientIP" url="http://localhost/flex/test/GetIp.aspx"
            result="onResult()"  method="GET"  showBusyCursor="true"/>
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.rpc.events.ResultEvent;
            //检测登录是否成功
            public function CheckUserProc():void
```

图 1-2　片段 MXML

3．ActionScript

ActionScript 是用于开发 Flex 的编程语言，它是一种强类型化、面向对象的编程语言。ActionScript 类似核心的 JavaScript 编程语言，基于 JavaScript 标准（ECMAScript4）。Flex 开发者使用 ActionScript 来描述客户端逻辑。例如，开发者使用 ActionScript 来定义事件监听器及句柄，设置或获取组件属性的值及处理回调函数等。目前最新版本为 ActionScript 3.0。

4．Flex 类库

Flex 类库包括 RIA 的类库，其中包含了 Flex 组件（容器与控件）、数据绑定、行为及其他功能。Flex 类库为开发者提供一致的视觉提示、交互模式和应用程序导航惯例。

5．Flex 组件

基于组件的模型使 Flex 应用程序的开发简化。开发者可利用 Flex 中包含的内置组件和扩展组件来添加新的属性及方法，创建新的组件。Flex 组件非常灵活，可为开发者提供大量控件，对组件外观、如何响应用户交互、文本的字体与字号、运行时的大小以及很多其他特征进行控制。

Flex 组件具有下列特点。

事件 应用程序或需要组件反应的用户操作。

行为 由应用程序或用户操作触发的可以看见或听见的变化。

纹理 控制组件外观的图形。

样式 各种特点的集合，如字体、字号及文本的对齐等。

尺寸 组件的高度与宽度（所有的组件都有默认的尺寸）。

Flex 类库提供两种类型的组件：容器和控件。开发者在使用 Flex 创建应用程序时，使用控件与容器描述用户界面。控件是一种用户界面组件，处理用户互动操作及显示可供用户直接通过该控件处理的数据，像 DataGrid 与 TreeControl。容器定义了 Flash Player 绘图表面的区域，控制容器内所有内容的布局，包括其他的容器与控件。容器包括用于数据输入的表单容器、对话框及网格等。

Flex 组件具有下列特点。

MXML API 用于声明控件及其属性值和事件等。

ActionScript API 用于调用控件的方法及在运行时设置其属性、事件。

可利用样式、纹理、字体来定制外观与视觉效果。

6．Flex 行为

Flex 类库也提供内置的行为，可以使开发者在应用程序中容易地添加动画和声音等来为用户的操作提供相应的环境。例如，开发者可以使用行为让对话框在收到焦点时轻微弹起。

行为是触发器与效果的结合体。触发是一个操作，如鼠标单击组件或是组件显示出来。效果是组件在一段时间内（单位为毫秒）可见的变化。效果的例子有淡入淡出、移动、改变大小、模糊、暂停或擦除过渡等。

开发者可以为单一触发定义多重效果，也可以按特定应用程序的需要定制效果或合成效果。

1.2 Flex 3.0

自从 Macromedia 推出 Flex 的第一个版本后，Flex 的发展速度令人惊讶。2004 年 3 月 Macromedia 正式推出 Flex 1.0 和 Flex Builder 1.0。现在看来，这两个产品还是很不成熟的。事实上，当时 1.0 的推出并没有引起很多人的关注，究其原因主要有两点：一是价格非常昂贵；二是运行环境只支持 J2EE，需要 Java 的服务器环境才能运行。对于大部分的 Flash 程序员而言，Java 的学习门槛太高，特别是没有网络编程经验的程序员，很难接受一门学习周期很长的新语言。

当然，软件本身的不成熟也是一个重要的原因。Flex Builder 1.0 的界面极其类似于网页编辑工具 Dreamweaver，用户很容易上手。但用户普遍反馈编辑工具的速度太慢，最后生成的 SWF 文件体积大。不过，Flex 的特性还是给很多人留下深刻的印象，用纯粹的 XML 描述语言也可以生成 SWF，这让很多人都觉得很神奇。

不久 Macromedia 便发布了升级补丁。2004 年 11 月，Macromedia 对 1.0 进行更大规模的

升级，推出 1.5 版本。这次的升级修正了上一版本中的很多错误，同时增加了一些新功能，使得 Flex 初具 RIA 的气质，主要表现在以下几方面。

支持运行时的共享库，开发人员利用此功能，可集中管理各种应用程序共享的资源。

提供了一组 Chart（图表）组件。

改进了布局功能。

更灵活地修改组件的样式和外观。

性能提高，运行速度提高了近 50%。

新增了一些实用的组件。

Flex 1.5 在运行性能上的提高让很多开发者看到了 Flex 的实用价值，这为 Flex 技术的推广起到很好的推动作用，也给下一版本打下坚实的基础。2006 年，Adobe 公司发布了 Flex 2.0，主要由以下技术和产品组成。

ActionScript 3.0 用于为 Flex 应用程序添加动态行为，它是基于 ECMAScript 的一种实现，类似于 JavaScript，将性能和开发效率作为首要目标。

Flex Framework 2.0 这是 Flex 运行时的核心，也称为 Flex SDK 2.0，是基于 MXML 和 ActionScript 的应用开发框架。提供了一套丰富的可扩展的用户界面组件、一个用来控制布局和用户交互的灵活模型以及一个功能强大的基础架构。

Flex Builder 2.0 一个在开放式平台 Eclipse 的基础上编写，并集合了 Flex Framework 和 ActionScript 的功能强大的开发工具。

Flex Data Services（FDS） 和 Flex 配合使用的数据服务器软件，提供了企业级的数据服务和即时通信功能。

Flex Charting Components 2.0 强大的图表组件。

Flash Player 9.0 改进虚拟机的脚本运行，包含 AVM2（ActionScript Virtual Machine）的 ActionScript 虚拟机和 ActionScript 3.0。速度快，支持运行时报错，遵循业界标准的调试方式，执行 ActionScript 的效率比以前高出 10 倍，并兼容早期版本。

另外，从 Flex 2.0 开始，Flex Framework 向用户免费开放，包括开发和编译 Flex 应用程序所需的完整命令行编译器，还有 Flex 组件库的所有源代码。从技术角度来看，Flex 2.0 的引入能使开发者构建 RIA 的新功能。主要体现在：ActionScript 3.0 采用了一种可以进行更强的编译时类型检查的编译模式，它完全支持 E4X（ECMAScript for XML）标准；Flex Builder 2.0 是在开源 Eclipse 框架的基础上构建的，包括了 Flex Framework 和命令行编译器；FDS 是一个增强型的服务器端架构，提供即时通信、大量数据传递等功能，支持 Remoting 通信方式，用户通过它可以直接访问 Java 对象。

2008 年，Adobe 再次对 Flex 进行升级，推出 Flex 3.0。Flex 3.0 包括 Flex 框架（也称为 Flex 类库）、Flex 命令行编译器、Flex 调试器、ASDoc 实用工具和调试器版本的 Flash Player。使用 Flex SDK 可以开发、编译和部署 Flex 应用程序，这些应用程序将连接到 XML 和 SOAP Web 服务，或者连接到各种服务器技术，例如使用 PHP、ColdFusion、Java 和.NET 的服务器技术。

下面列出 Flex 3.0 的一些主要新特性和增强功能。

Adobe AIR 的本机支持

Flex 3.0 增加了对 Adobe AIR 的支持，后者允许开发人员在 HTML、Ajax、Flash 和 Flex

中使用现有的 Web 技术构建 RIA 并将其部署到桌面系统中。Flex 3.0 引入了一些新组件，并将 Adobe AIR 开发工具包含到 SDK 和 Flex Builder 中。

持久性框架缓存

在对 Adobe 平台组件使用新的 Flash Player 缓存时，可使 Flex 3.0 应用程序的大小减少到 50K。

Advanced DataGrid 组件

Advanced DataGrid 是一个新组件，用于将请求的特性添加到 DataGrid 中。例如对分层数据的支持和基本透视表功能，目前仅可在 Flex Builder Professional 中使用。

OLAP DataGrid 组件

OLAP（在线分析处理）网格数据允许以紧凑格式聚合数据，并用由行和列组成的二维网格显示这些数据聚合。目前仅可在 Flex Builder Professional 中使用。

Enhanced Constraints 布局机制

Enhanced Constraints 构建在已有的基于约束的布局机制之上，允许使用兄妹关系约束（在 Flex 2.0 中只可以定义父子约束）创建复杂的、可重新调整大小的布局。

Flex Charting 包增强

Flex 3.0 通过许多增强功能改进了 Charting 包。坐标系统现在只可以支持几个轴，DateTimeAxis 允许执行工作周过滤。新的面向数据的图形 API 允许绘制数据坐标，用图表呈现适当屏幕位置上的一切内容。对于现有所有图表，还有一些新格式选项和新添加的交互功能。只可与 Flex Builder Professional 一起使用。

Flex Component Kit for Flash CS3

Flex Component Kit for Flash CS3 提供了用于创建可无缝集成到 Flex 应用程序中的 Flash 内容的完整工作流。Flash 用户现在可以使用熟悉的 Flash 时间轴模型开发组件，然后通过使用少许简单的模式使 Flex 开发人员能够引入这些组件，而无须添加额外代码。Flex Component Kit for Flash CS3 需要额外下载，可以从 Adobe 网站获得。

Flex Ajax Bridge 库

Flex 3.0 现在包括 Flex Ajax Bridge、一个可插入 Flex 应用程序的小型代码库、一个 Flex 组件或一个空 SWF 文件（可以导出该文件实现浏览器中的脚本编写）。使用 Flex Ajax Bridge 可以使 ActionScript 类可用于 JavaScript，无须编写任何额外的编码。在插入库之后，就可以使用 JavaScript 实现用 ActionScript 实现的任何操作。

Flex 开源

Adobe Flex 现在是开源的，包含用于框架、编译器、调试器以及更多可在 Mozilla Public License 下使用的工具的源代码。更多信息可以访问 http://opensource.adobe.com/flex。

1.3 MXML 概述

MXML 是一种用于创建用户界面的功能强大的标记性语言。MXML 文件将 Flex 项目中的所有文件组合在一起，形成一个有机的整体。而在编译时 MXML 会转化成 ActionScript。因此，可以简单地将 MXML 结构理解成一种动态的 ActionScript 类。

1.3.1　MXML 命名规范

MXML 是一种 XML 语言，可以使用它去布置 Adobe Flex 应用程序的用户界面。还可以使用 MXML 去定义其他的方面，如存取服务器端的数据，将用户组件与服务器端数据源进行绑定等。

MXML 看起来与所熟悉的 HTML 很类似。然而，MXML 更为结构化，并提供更为丰富的标签集。MXML 与 HTML 之间最大的不同之处在于，MXML 所定义的应用程序将被编译成 SWF 文件并由 Flash Player 进行渲染，提供比 HTML 程序更丰富的和动态的用户界面。

MXML 对大小写敏感，而且文件名和变量名都区分大小写。字母大小写错误是编程中常见的错误，隐蔽性较高，有时很难排错。所以，建议读者在编写代码时，应该采用合理的命名规范，避免出现这类错误。

在 Flex 程序中，每个 MXML 文件都必须以小写的 mxml 作为后缀，文件名要遵循 ActionScript 中变量的命名规则。因为在程序中，所有的 MXML 文件都被视为用户自定义的组件，相当于一个对象，可以使用代码动态创建。也就是说，MXML 文件可以直接被 ActionScript 当作一种用户定义的数据类型来使用。因此，MXML 文件不能和 ActionScript 类文件同名，否则就会造成类型冲突。

在 ActionScript 中，变量名称必须以字母或下划线开始，且只能包含字母、数字和下划线。需要注意的是，MXML 文件不能命名为 Application。因为，Application 是主程序文件所采用的默认标记，不可以再被标记，也不能和程序中任何一个组件的 id 值同名。

例如，下面是 MXML Application 文件 HiFlex.mxml 的源代码。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
    <mx:Label x="27" y="44" text="Welcome to Flex world.."
        id="lblStr" fontSize="21" color="#FF001E"/>
</mx:Application>
```

现在将 Label 组件的 id 属性尝试设置为与文件名相同的 HiFlex，如下所示。

```
<mx:Label x="27" y="44" text="Welcome to Flex world.."
        id="HiFlex" fontSize="21" color="#FF001E"/>
```

此时，如果编译将无法通法，会看到提示为 identifier and class may not have the same name. 的错误提示信息。说明，组件的唯一 id 属性不可以与对象名相同。

另外，还需要注意的是程序中不可以使用 mx 作为目录名。因为 mx 是 Flex Framework 官方组件的命名空间，受到 Flex 编译器的保护。假设，在程序中创建了名为 mx 的目录，这个目录中存放的 MXML 文件和 ActionScript 类文件都将无法使用，将会被编译器忽略。

为了避免文件名冲突，同时也为了增强程序的可读性，文件和变量一般应该采用有意义的单词、名词简写或字符组件来命名。下面列出了一些命名建议。

为了增加程序的可读性，可在前面加一个表示其类型号的前缀，名称也尽量使用能代表该变

量用处的单词。例如，strName、intCount、pnlLogin、btnSend 和 txtUserPwd 等。
使用下划线组合单词，例如，User_Name、User_Email、Product_List、Move_Speed 等。
将常量全部使用大写，例如圆周率用 PI 表示，单击事件名用 CLICK_EVENT 表示。

1.3.2 MXML 文件结构

MXML 文件其实就是一个标准的 XML 文件，因此可以参照 XML 语法来分析 MXML 文件的内容。

这里以上节 HiFlex.mxml 文件的代码为例。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
    <mx:Label x="27" y="44" text="Welcome to Flex world.."
        id="lblStr" fontSize="21" color="#FF001E"/>
</mx:Application>
```

在 XML 文件中，需要在第一行声明 XML 文件采用的语法版本号和文件采用的编码格式。从上面的代码中可以看到这两个属性：version 和 encoding，其中 encoding 编码格式这个属性是可选的。选定的编码格式必须符合采用的编码格式，默认为 utf-8。在中文操作系统中，一般使用 utf-8 这个编码格式，它兼容采用双字节编码的语言和其他常见的西方语言，而且可以跨平台，也是使用最广泛的编码格式。另外，也可以使用简体中文的专用编码格式 GB2312。

mx:Application 标签是一个特殊的标签。在每个 Flex 项目中，可能有多个 MXML 文件，但作为程序入口的运行文件只有一个，即主文件。其标识是根节点为 mx:Application。另外，在一个 MXML 文件中只能出现一个 mx:Application 标签。

在 mx:Application 标签中，还看了一个属性：xmlns:mx="http://www.adobe.com/2006/mxml"，表示将 mx 定义为 XML 的命名空间。xmlns 标签专门用来定义 MXML 文件的命名空间，XML 命名空间可以用来定义一套独立的 XML 标签，并且为这些标签指定特殊的解析方式。例如，XML 中默认的标签格式为<Button>node</Button>，这里 Button 节点被看作一个普通文本节点，没有其他特殊意义。定义命名空间后，在节点上加上空间前缀<mx:Button></mx:Button>，这时 mx:Button 就代表 mx 命名空间下的 Button 对象。

mx 命名空间对应的路径是 http://www.adobe.com/2006/mxml，Flex 的配置文件将这个路径定义为一个全局资源标识符，并对应了一个 XML 文件。在这个文件中，列出了 mx 这个命名空间下的所有标签。在 Flex Builder 3 的安装路径下，进入 sdks\3.0.0\frameworks 目录中，找到 flex-config.xml 文件并使用记事本打开，会看到如下所示的内容。

```
<namespaces>
<!-- Specify a URI to associate with a manifest of components for use as
MXML -->
<!-- elements.-->
   <namespace>
```

```
      <uri>http://www.adobe.com/2006/mxml</uri>
      <manifest>mxml-manifest.xml</manifest>
    </namespace>
</namespaces>
```


从上面的配置中发现 http://www.adobe.com/2006/mxml 这个 URI 和 mxml-manifest.xml 文件对象。下面打开同目录下的 mxml-manifest.xml 文件。在该文件中列出了 MXML 中的所有标签和与标签关联的文件路径。下面列出的是其中部分内容。

```
<?xml version="1.0"?>
<componentPackage>
    <component id="HTML" class="mx.controls.HTML"/>
    <component id="Window" class="mx.core.Window"/>
    <component id="Accordion" class="mx.containers.Accordion"/>
    <component id="Application" class="mx.core.Application"/>
    <component id="ApplicationControlBar" class="mx.containers.Application
    ControlBar"/>
    <component id="Array" class="Array" lookupOnly="true"/>
    <component id="Button" class="mx.controls.Button"/>
    <component id="Canvas" class="mx.containers.Canvas"/>
    <component id="CheckBox" class="mx.controls.CheckBox"/>
    <component id="Image" class="mx.controls.Image"/>
    <component id="Object" class="Object" lookupOnly="true"/>
    <component id="Panel" class="mx.containers.Panel"/>
</componentPackage>
```

在上述给出的代码中，节点的 id 属性表示标签名称，class 属性表示类文件的路径。例如，mx:Application 标签也就是对应了 mx.core.Application 类。

在开发中，当程序中有很多的 MXML 文件和 ActionScript 文件时，为了方便调用，可以将功能相似的文件放在一个文件夹中。然后再定义一个命名空间，这样会节省很多时间。在自定义命名空间时，为了方便，一般直接指定命名空间包括的标签路径，如：

```
xmlns:MyComp="components.*"
```

因为使用了通配符"*"，components 目录下的所有 MXML 文件和 ActionScript 类文件（不包括目录和目录中的文件）都被包括在 MyComp 命名空间下。假设，components 中有一个 LoginPanel.mxml 文件，则在程序中调用这个文件时，可以使用如下代码：

```
<MyComp:LoginPanel></MyComp:LoginPanel>
```

这样，MyComp 下的标签会自动被指向 componets 中的文件。当标签数量较多且分布在不同文件夹中时，可以模仿 Flex 配置文件的做法，使用 XML 文件来描述标签的路径。在本书后面的内容中，会经常看到命名空间的使用。

现在回到 MXML 文件，Application 标签中还一个属性 layout，这个属性定义了 Application

节点下元素的布局方式。由于 Application 是根节点，因此它的布局方式决定了程序的总体布局方式。absolute 属性值表示绝对定位，这样文件中的所有元素将按照各自的 x、y 坐标来定位。

最后，看一下 Application 节点中的内容。这里仅包含了一个 mx:Label 节点，代表一个 Label 组件，组件中的属性还定义了初始化时一些状态，像坐标、字体大小、颜色以及显示的文本等。

1.4 ActionScript 3.0 概述

ActionScript 是针对 Flash Player 运行时环境的脚本编程语言，它使 Flash 应用程序实现了交互性、数据处理以及其他许多功能。Flash Player 中内置的 ActionScript Virtual Machine（AVM）执行 ActionScript。ActionScript 代码通常被 Flash 提供的编译器编译成“字节码格式”（一种由计算机编写且能够为计算机所理解的编程语言）。字节码嵌入 SWF 文件中，这种字节码文件由运行时环境 Flash Player 执行。

1.4.1 ActionScript 3.0 简介

ActionScript 1.0 最初随 Flash 5 一起发布，这是第一个完全可编程的版本。在 Flash 7 中引入了 ActionScript 2.0，这是一种强类型的语言，支持基于类的编程特性，比如继承、接口和严格的数据类型。Flash 8 进一步扩展了 ActionScript 2.0，添加了新的类库以及用于在运行时控制位图数据和文件上传的 API。通过使用新的虚拟机 ActionScript Virtual Machine 2（AVM2），Flash CS3（附带 ActionScript 3.0）大大提高了性能。

ActionScript 3.0 也是一种功能强大的面向对象编程语言，它是 Flash Player Runtime 演化过程中的一个重要阶段。ActionScript 3.0 是一种适合快速构建效果丰富的互联网应用程序（RIA）的语言。图 1-3 中演示了 ActionScript 3.0 的执行顺序及各部分的作用。

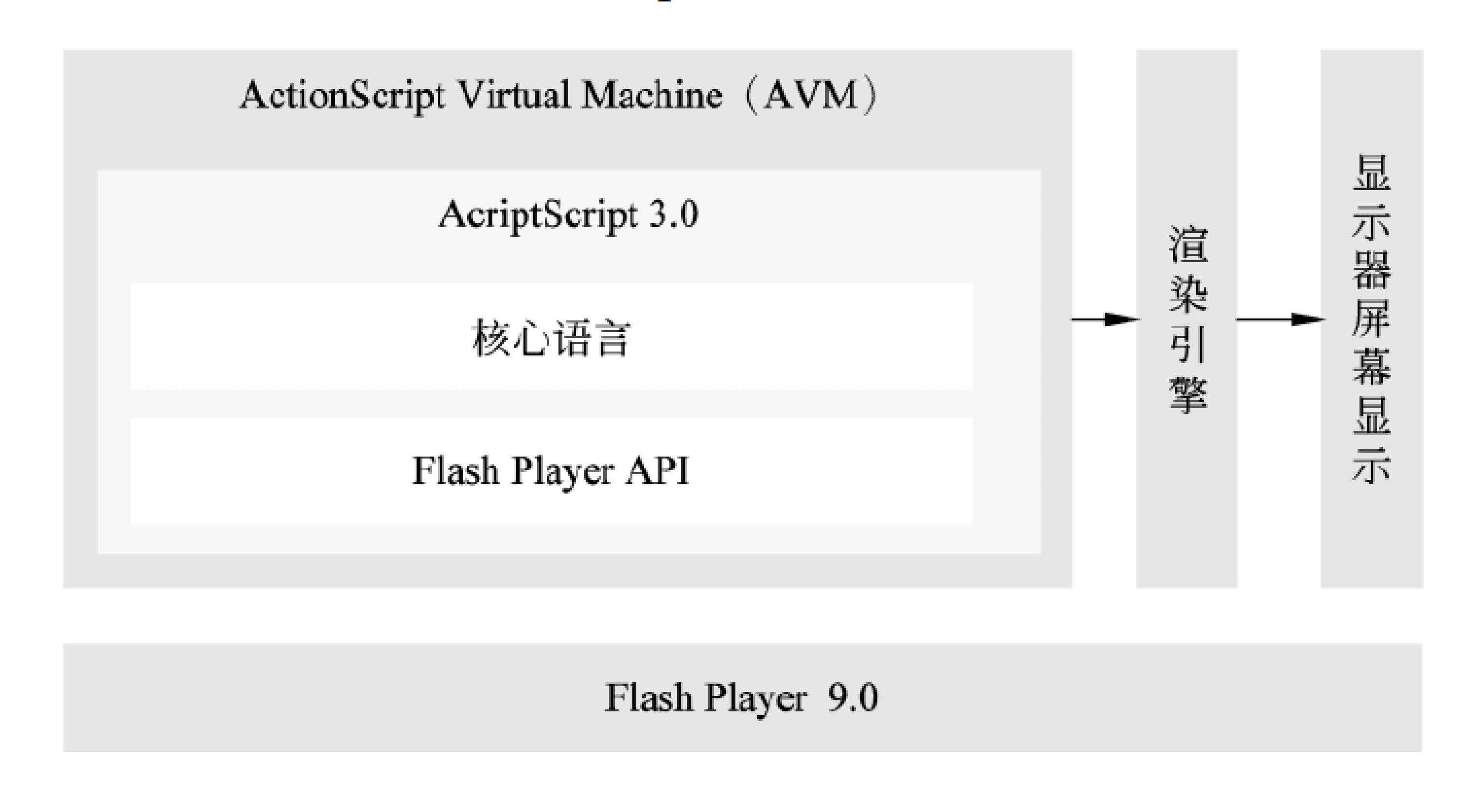


图 1-3 ActionScript 3.0 执行顺序

ActionScript 的老版本（ActionScript 1.0 和 2.0）提供了创建效果丰富的 Web 应用程序所

需的功能和灵活性。ActionScript 3.0 现在为基于 Web 的应用程序提供了更多的支持。它提供了出色的性能、简化了开发的过程，因此更适合于高度复杂的 Web 应用程序和大数据集。



用 ActionScript 1.0 或 2.0 编写的 SWF 文件无法加载用 ActionScript 3.0 编写的 SWF 文件。

ActionScript 3.0 符合 ECMAScript Language Specification 第三版。它还包含基于 ECMAScript Edition 4 的功能，像类、包和名称空间，可选的静态类型，生成器和迭代器，以及非结构化赋值（Destructuring Assignments）。随着 Web 应用程序项目需求的增加，也要求 ActionScript 引擎有重大的突破。ActionScript 3.0 引入了新的高度优化的 ActionScript Virtual Machine 2（AVM2）。与 AVM1 相比，AVM2 的性能有了显著的提高。这使 ActionScript 3.0 代码的执行速度几乎比以前的 ActionScript 代码快了 10 倍。为了向后兼容现有的内容，Flash Player 将继续支持 AVM1。

与 ActionScript 2.0 相比，ActionScript 3.0 采用了一种可以进行更强的编译时类型检查的编译模式，它结合了多种语言（像 Java 和 C#）的优势。在 XML 数据的处理上，遵循 E4X 标准，ActionScript 3.0 自带的 XML 类，是一种直接扩展了 E4X 标准的数据类型。开发者可以很方便地解析和架构 XML 数据。同时，ActionScript 3.0 添加了正则表达式支持，为开发者提供了处理复杂字符串的技术支持。

ActionScript 3.0 在很多功能方面都有了很大的改进，例如增强处理运行时错误的能力，更全面地面向对象，支持二进制数据处理，支持 Socket 通信，提供 Flash Player API 等。可以说，ActionScript 3.0 已经成为一个遵循业界标准、完全面向对象、表现优异的编程语言。

1.4.2　在 Flex 中 ActionScript 的使用方式

前面讲过，MXML 用于为 Flex 应用程序进行用户界面组件布局，它属于表示层，最终要编译成 ActionScript，并生成 ActionScript 类文件在 Flash Player 上运行。这一点对于 Java 开发者来说可能很容易理解，MXML 就好像是 JSP、Struts、JSF，它们最终都会编译成 Java 类文件，并在具备 Java 虚拟机环境的浏览器上运行。

所以说，Flex 最核心的还是 ActionScript。在 Flex 中，ActionScript 是以类库的方式出现的，该类库包含组件（容器和控件）、管理器类、数据服务类和所有其他功能的类。本节主要介绍在 Flex 中使用 ActionScript 的方法。

在 Flex 中使用 ActionScript 主要有 3 种方式：内联方式、级联方式和外联方式，下面依次介绍。

1. 内联方式

这种方式直接将 ActionScript 代码作为事件的属性值。通常都是仅有一行，相对简单，如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
    <mx:Label x="27" y="44" text="Welcome to Flex world.."
        id="lblStr" fontSize="21" color="#FF001E"
         click="mx.controls.Alert.show('Hello Flex.')"/>
</mx:Application>
```

2. 级联方式

这种方式将 ActionScript 代码放入<mx:Script></mx:Script>代码块中，然后将方法作为事件的属性值。这样可以在一个文件中重用 ActionScript 代码，而且可以在调用方法时传递参数。

<mx:Script>标签可以放置在根节点下的任何位置，但是必须使用 CDATA 将代码包起来。CDATA 标签是 XML 中特殊字符的专用标签。XML 解析器通常情况下会处理 XML 文档中的所有文本。如果在 XML 文档中使用类型的“<”，那么解析将会出现错误。因为，解析器会认为这是一个新元素的开始。还有，在 XML 中“<”和“&”字符是禁止使用的，它们和正常的 XML 标签冲突，会导致 XML 文件不能正确解析。使用 CDATA 标签可以很好地解决这个问题，在 CDATA 内部的所有内容都会被解析器忽略。

下面给出对上例修改后采用级联方式的代码。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
    <mx:Label x="27" y="44" text="Welcome to Flex world.."
        id="lblStr" fontSize="21" color="#FF001E"
         click="ShowMsg('Flex')"/>
         <mx:Script>
            <![CDATA[
                import mx.controls.Alert;
                private function ShowMsg(msgstr:String):void{
                    Alert.show("Hello "+msgstr);
                }
            ]]>
         </mx:Script>
</mx:Application>
```

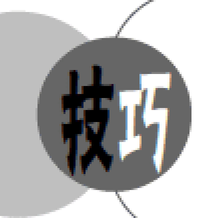
从上述代码中可以看出，将 MXML 和 ActionScript 代码分开，文件结构清晰，更容易维护。编写代码时，添加程序注释是一个很好的习惯，可以增强程序的可读性。在编译时，注释部分被自动跳过，被注释的代码也不会执行。ActionScript 中有如下两种添加注释的方式。

注释单行使用“//程序注释内容”。

注释多行使用“/*程序注释*/”。

另外，MXML 文件也可以添加注释，方式与 XML 中相同，格式如下：

```
<!-- MXML 注释 -->
```



当程序的代码量很大时，添加适当的注释非常有必要。简短扼要的注释，往往可以指出程序中的关键点，使程序员在编写过程中时刻掌握程序的结构。

3. 外联方式

如果在 MXML 文件中插入的 ActionScript 代码量很大，这时候会考虑将代码提取出来，放在一个单独的 ActionScript 文件中。这样，可以将 ActionScript 代码和 MXML 界面分离，让程序变得更易于维护。

这里指的是外部的 ActionScript 文件，而不是 ActionScript 类文件。虽然两者都是以.as 作为文件名后缀，但区别很大。前者只是将 MXML 文件<mx:Script>标签中的代码提出来，放在一个.as 文件中；而后者则是 ActionScript 类文件，遵循面向对象的语法。具体的区别将在本书后面介绍。

要将 MXML 文件中的 ActionScript 代码关联到一个外部文件，首先需要在 MXML 文件中进行设置，指定文件的路径。这需要使用<mx:Script>标签来完成，格式如下：

```
<mx:Script source="外部文件路径" />
```

这里 source 属性即为外部文件的路径。在编译时，文件中的代码被复制到程序中执行，和直接写在<mx:Script>标签中的代码功能相同。

例如，这里要将 ActionScript 代码关联到文件 MyTool.as。首先在 MXML 中进行设置，如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
    <mx:Label x="27" y="44" text="Welcome to Flex world.."
        id="lblStr" fontSize="21" color="#FF001E"
         click="ShowMsg('Flex')"/>
         <mx:Script source="MyTool.as" />
</mx:Application>
```

然后，在相同目录中创建一个 MyTool.as 文件并输入如下所示的代码即可。

```
//演示关联外部 ActionScript 文件
import mx.controls.Alert;
private function ShowMsg(msgstr:String):void{
    Alert.show("Hello "+msgstr);
}
```

1.5 部署 Flex 3.0 开发环境

本节前面介绍了很多关于 Flex 的基础概念，本节将以最新的 Flex 3.0 为例详细介绍开发环境的部署。从 Flex 出现到现在，每个版本的变化都比较大，而且在一个大版本中，官方可能根据情况发布升级补丁，导致出现多个版本并存的情况。因此，使用时要注意，本书中 Flex 3.0 指的是 Adobe 第一次发布的 3.0 正式版，不包括任何升级包和补丁。

1.5.1 获取 Flex 3.0

Flex Framework 是 Flex 的核心，也是开发环境必备的。在 Flex Framework 中包含免费的类库，可以供用户独立安装。在 Adobe 官方网站 http://www.adobe.com 可以下载最新 Flex 3.0。

Flex Framework 中包含了 Flex 编译器，安装后，即可使用简单的文本编辑器、XML 编辑器或支持文本编辑的任何软件来编写 Flex 程序。然后再结合编译器，就可以手动进行 Flex 3.0 应用程序的开发。这种方式对开发者的要求比较高，要求对 MXML 语言和 ActionScript 非常熟悉，所以不推荐初学者使用。

任何一种编程语言的成功都离不开一个优秀的开发环境，Flex 也不例外。这里，建议读者使用 Adobe 的 Flex 集成开发环境——Flex Builder，其最新版本为 Flex Builder 3，同样可以在 Adobe 官方网站下载获取。

1.5.2 安装 Flex Builder 3

Flex Builder 3 不仅仅是旧版本的简单升级，它整合了 Flex Framework 3.0，并提供了一系列强大的功能，例如可视化的拖曳布局模式、完善的代码提示、所见即所得的 CSS 设计模式等。使用 Flex Builder 3 可以让开发者的效率得到显著提高。

在安装之前，读者首先需要了解 Flex Builder 3 对系统硬件和软件的配置要求。如下是官方推荐的运行环境：

CPU Intel Pentium 4 处理器。

操作系统 Windows XP with Service Pack 2。

内存 1GB。

硬盘 500MB 的可用硬盘空间。

Java 虚拟机 Sun JRE 1.4.2、Sun JRE 1.5 （Flex Builder 内置）、IBM JRE 1.5 或 Sun JRE 1.6。

插件安装要求 Eclipse 3.2.2 或更高版本、IBM Rational Software Architect 7.0。

其他　Adobe Flash Player 9。

下载后会得到一个可执行文件，直接双击开始安装。现在来看一下它的安装步骤。

（1）安装程序会将所需的文件保存到系统临时存储位置，来做安装准备。这些就绪之后会弹出如图 1-4 所示的选择语言对话框。



图 1-4　选择语言

（2）Adobe Flex Builder 3 发布时仅支持 English 语言，单击 OK 按钮继续。在进入的对话框中显示了本安装程序的功能介绍，如图 1-5 所示。

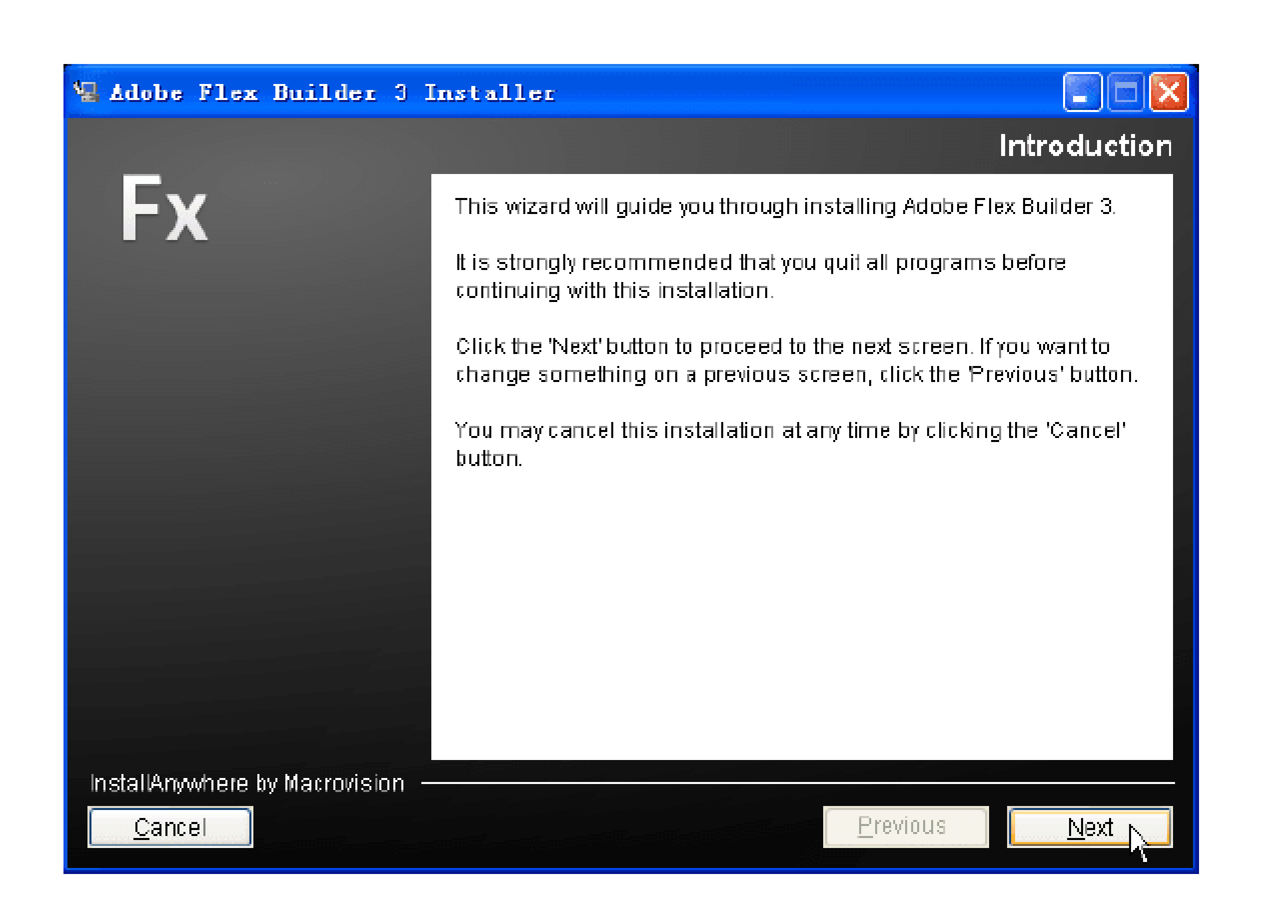


图 1-5　浏览功能

（3）单击 Next 按钮，进入安装过程的第一步，在这里列出了要继续所需的同意许可描述。启用 I accept the terms of the License Agreement 单选按钮后，单击 Next 按钮，如图 1-6 所示。

（4）我们会看到图 1-7 所示的界面，这里可以设置 Flex Builder 3 的安装位置。如果想要自己设定安装目录，可以单击 Choose 按钮选择要安装的目录。此处安装到 D:\Program Files\Adobe\Flex Builder 3 目录。

（5）单击 Next 按钮，选择要安装的插件。这里包括针对 Internet Explorer 浏览器和 Firefox 浏览器的 Flash Player 9，以及支持 ColdFusion 语言和 JavaScript 脚本的插件，如

图 1-8 所示。



图 1-6　阅读安装许可协议

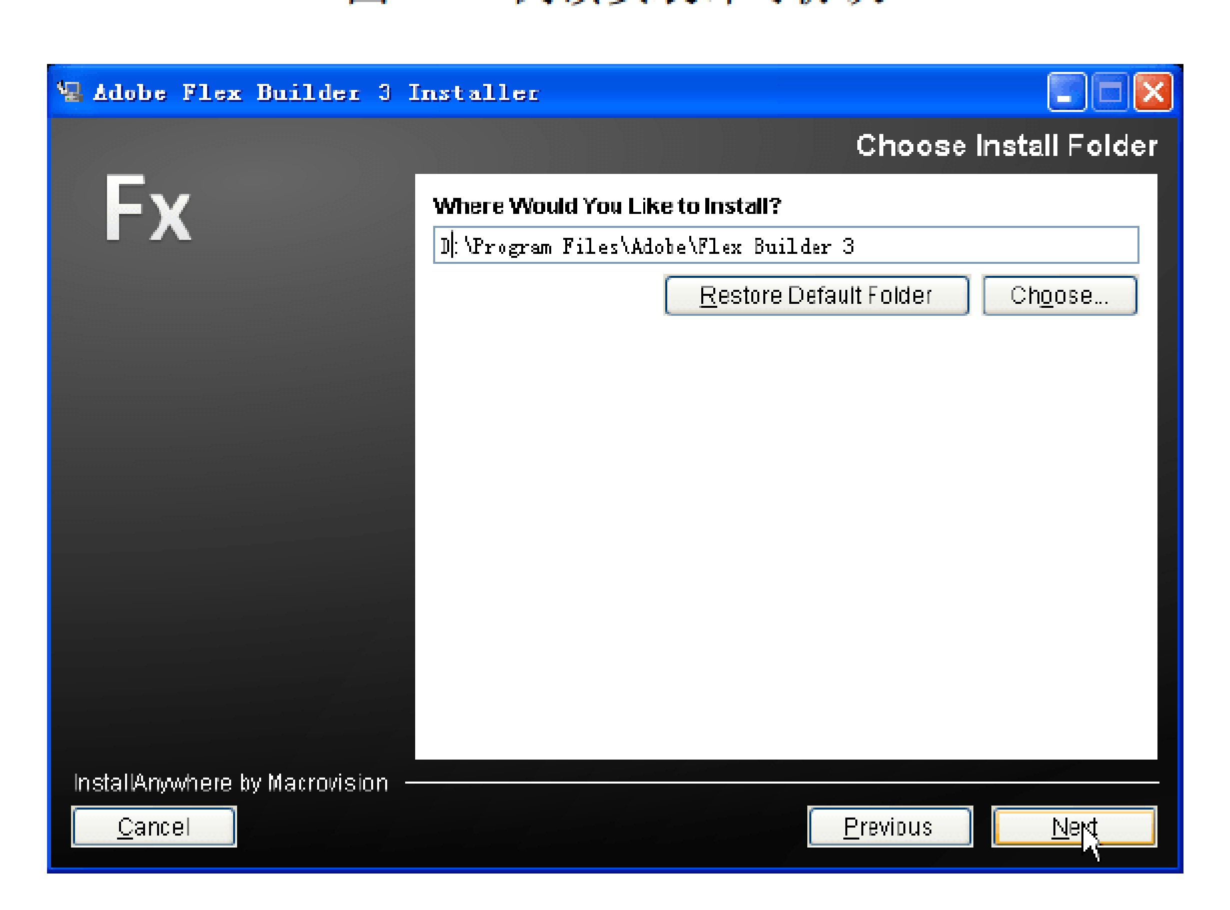


图 1-7　选择安装位置

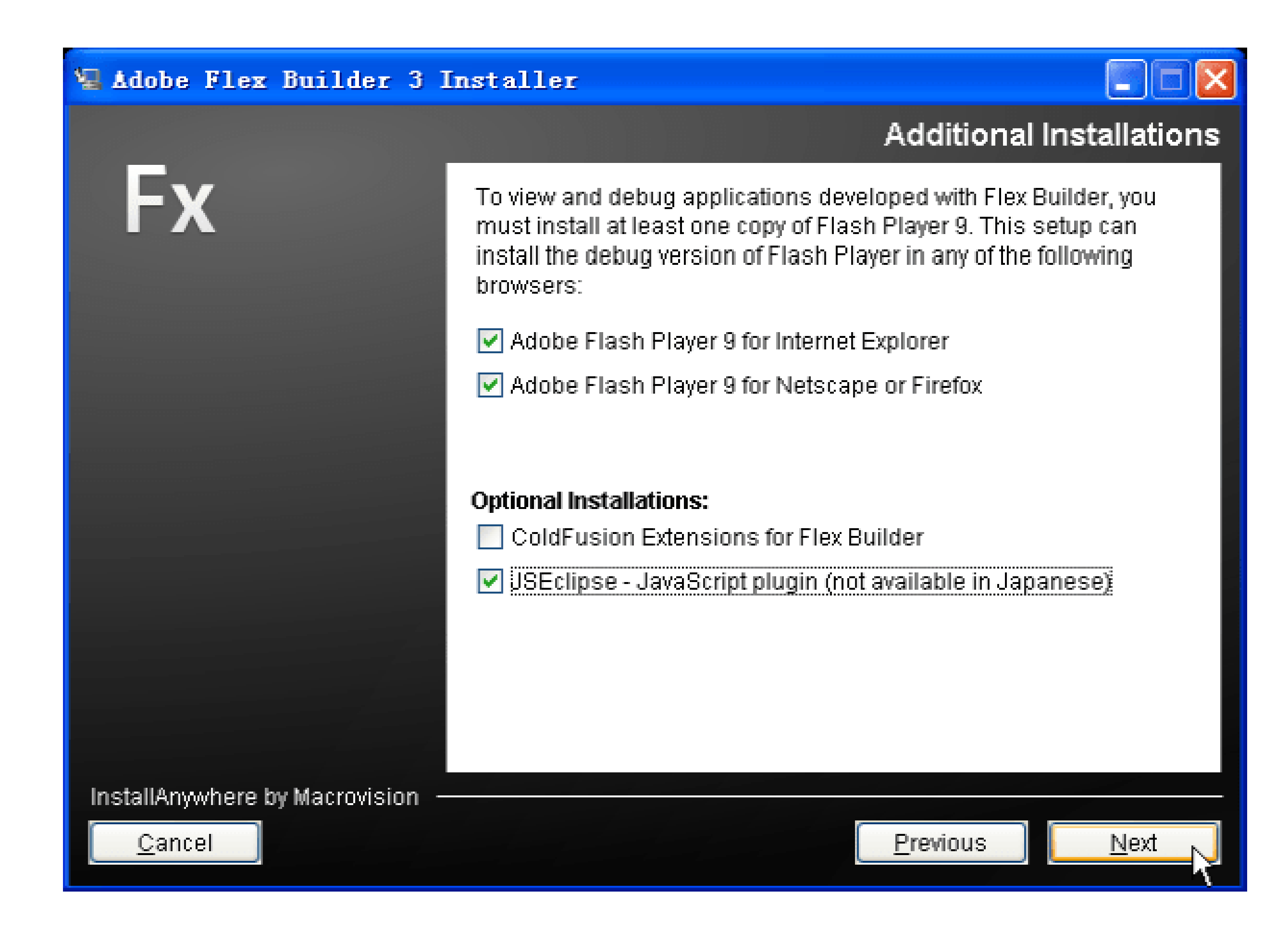


图 1-8　选择安装插件

这一步很重要，Flex 3.0 需要 Flash Player 9 或更高版本才能运行。因此这两项都必须启用。



建议读者安装 JSEclipse 插件，它是 Adobe 开发的 JavaScript 编辑器插件，用于方便地开发 JavaScript 脚本。

（6）单击 Next 按钮，浏览此前设置的安装概要，包括安装路径、插件设置以及所需空间，如图 1-9 所示。

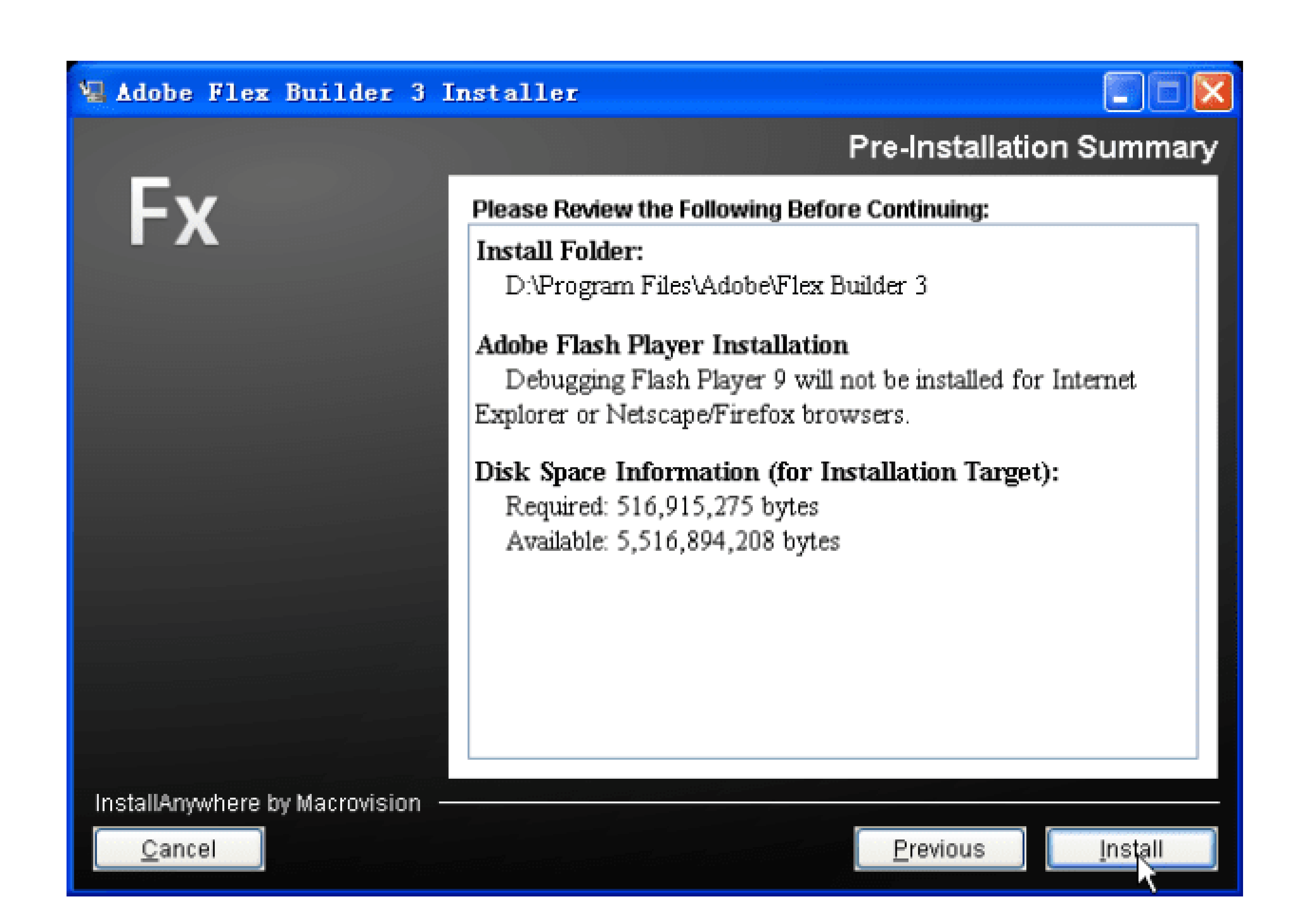


图 1-9　安装前的确认

（7）单击 Install 按钮，这样就会开始安装 Flex Builder 和 Flex 中开发 RIA 应用程序所需要的其他组件。复制文件和组件的过程长短与计算机的配置成正比。在安装过程中，会在下方显示将要安装的组件和组件的安装进度，如图 1-10 所示。



图 1-10　复制文件

（8）安装完成后会出现图 1-11 所示的 Install Complete 窗口，提示成功安装到指定位置。单击 Done 按钮结束安装程序，现在即可开始构建 RIA 应用程序。

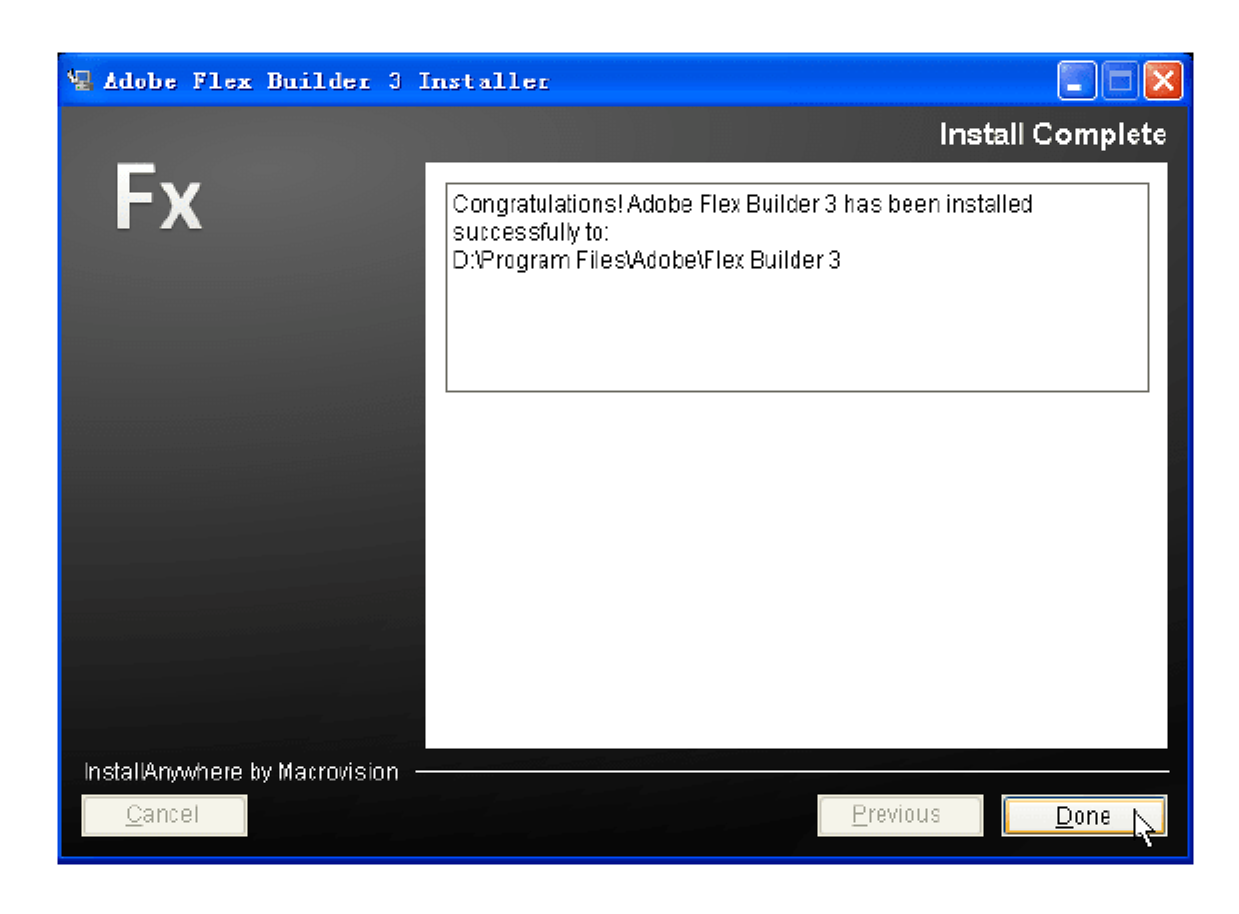


图 1-11　安装完成

1.5.3　第一个 Flex 程序

现在已经完成 Flex Builder 3 的安装，并且在安装过程中，Flex Builder 3 已经部署了开发 Flex 3.0 所需的各个组件。本节将通过创建一个简单的 Flex 程序实例，来说明如何创建项目、编译调试并发布最后的 SWF 文档。重点是帮助读者理解前面讨论的概念是如何在 Flex 中应用的。

（1）要创建一个 Flex 应用程序，必须首先创建一个 Flex 项目。Flex Builder 3 会创建所有所需的目录，并创建和管理运行项目所需的文件。启动 Flex Builder 3，在菜单中选择 File | New | Flex Project 命令。

（2）弹出 New Flex Project 对话框，在 Project name 文本框中指定 Flex 项目名称，这里输入 FirstFlexApp。默认会启用 Use default location 复选框，这样所有创建的项目都保存到这个目录下，如图 1-12 所示。

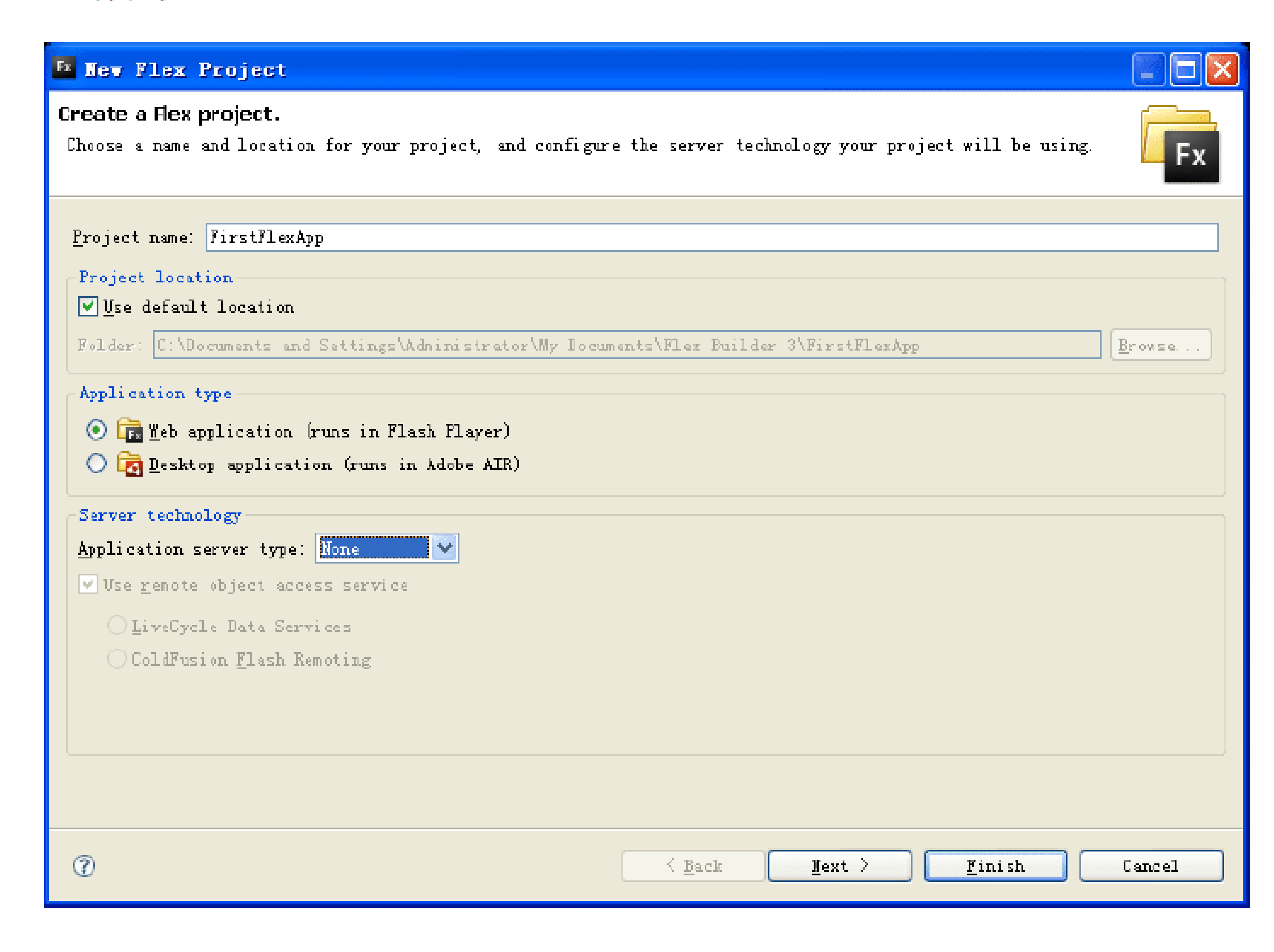


图 1-12　New Flex Project 对话框

在 Application type（应用程序类型）中的 Web application (runs in Flash Player)表示 Web 应用程序（运行在 Flash Player 里面），Desktop application (runs in Adobe AIR)表示桌面应用程序（运行在 Adobe AIR 中）。在 Application server type 下拉列表框中可以指定 Flex 使用的服务器端技术。

（3）单击 Next 按钮，在进入的 Configure Output 窗口中设置经过编译后 Flex 程序在本地的位置。推荐使用默认值，位于项目文件夹下，输入文件夹名称为 bin-debug，如图 1-13 所示。

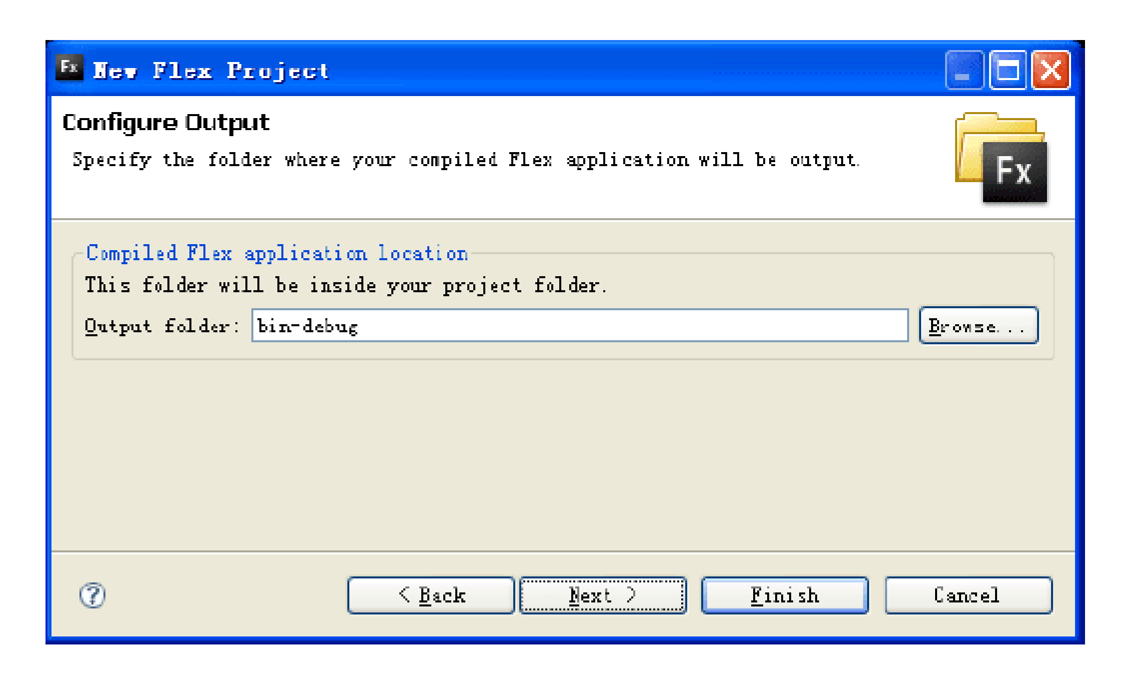


图 1-13　Configure Output

（4）单击 Next 按钮，设置源路径和类库路径，如果需要第三方类库、项目，或自己开发的类库、项目，可以单击相应按钮来完成，如图 1-14 所示。

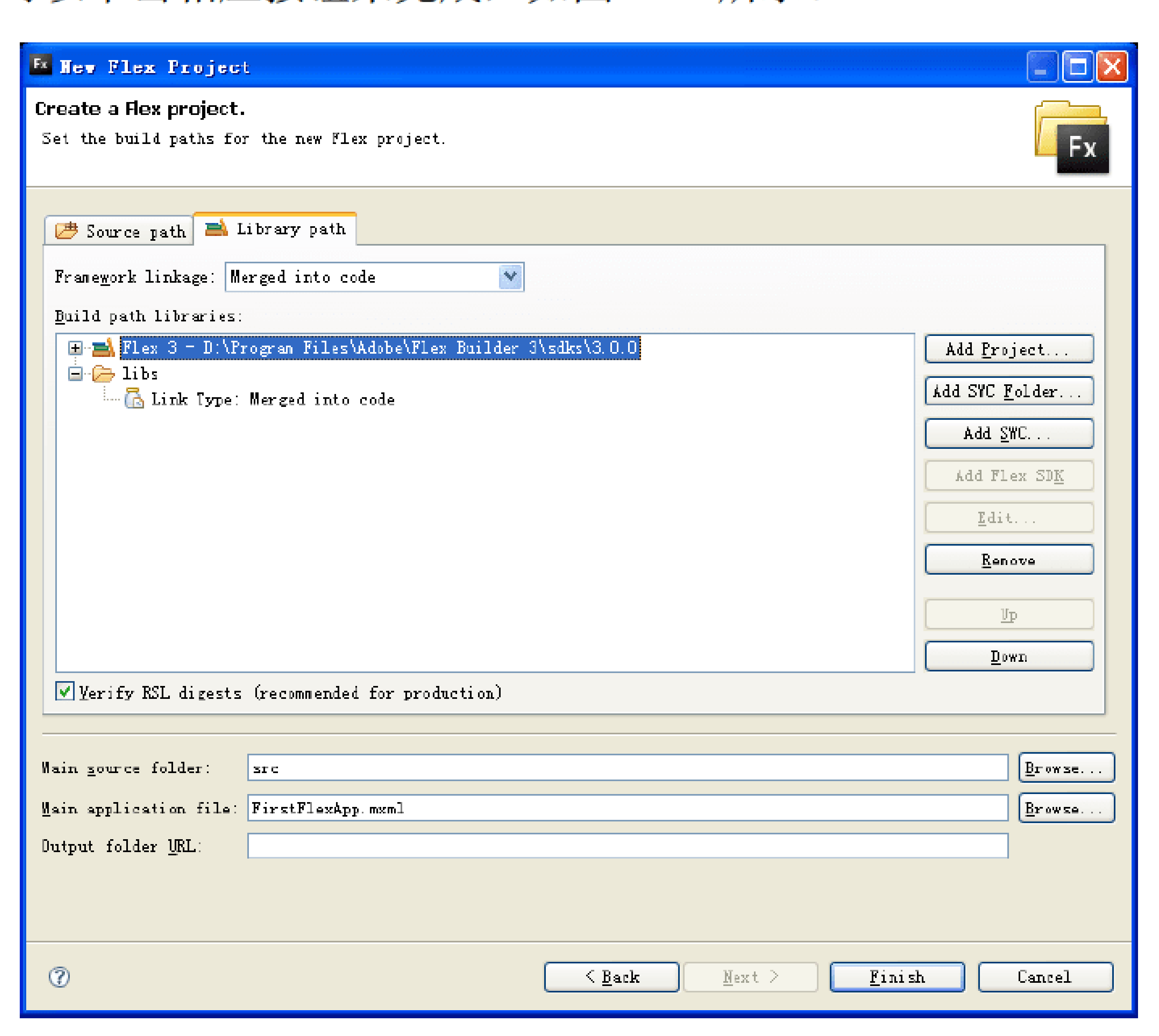


图 1-14　设置数据源路径和类库路径

（5）单击 Finish 按钮，完成 Flex 项目 FirstFlexApp 的创建过程。进入 Flex 程序的开发环境，并显示 FirstFlexApp.mxml 中的 MXML 代码，如图 1-15 所示。

（6）单击 Design 按钮进入 Design 视图。然后从 Components 窗格中展开 Layout 节点，拖曳一个 TitleWindow 组件到设计区域。

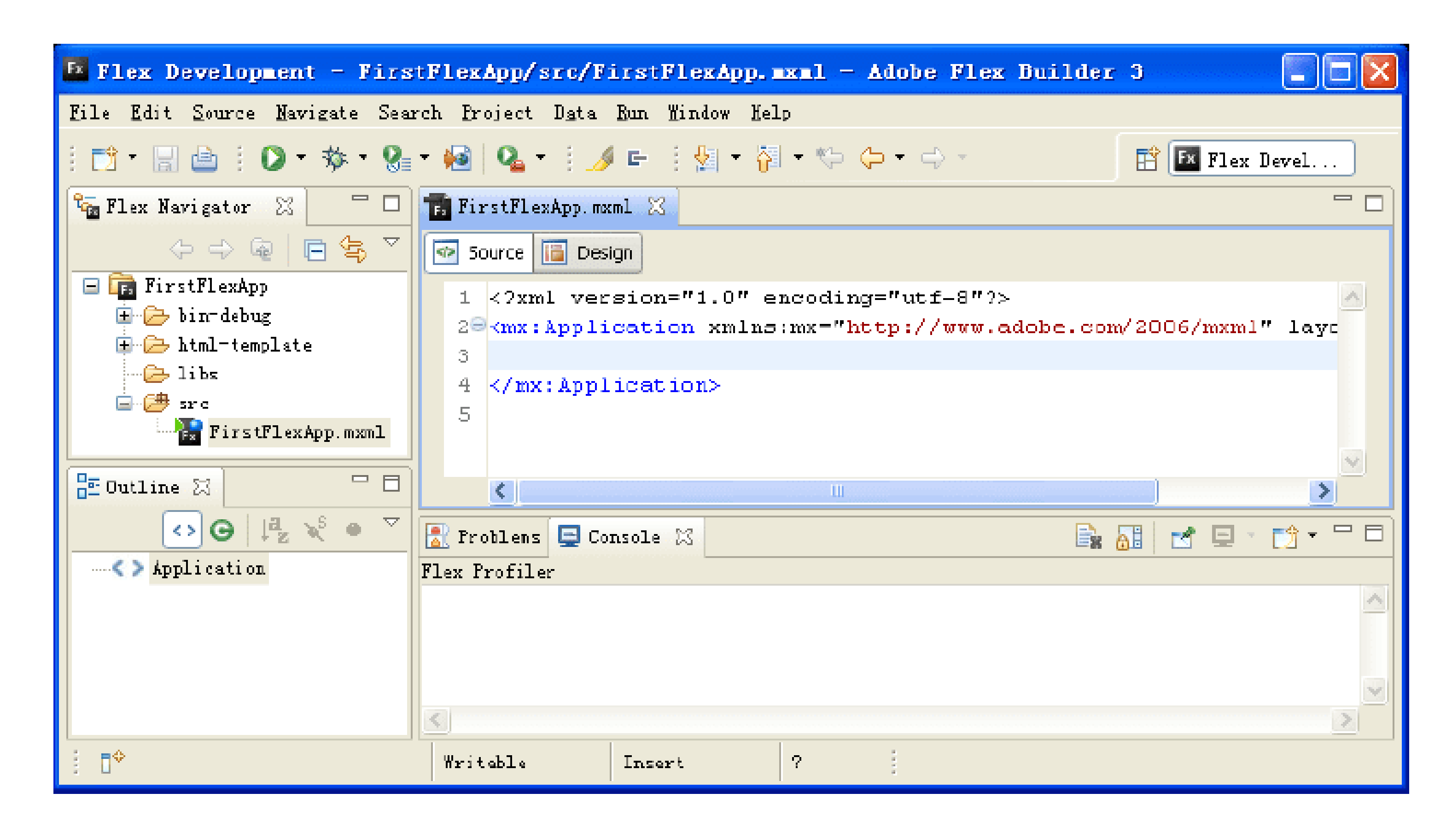


图 1-15　FirstFlexApp.mxml 内容

（7）在 Flex Properties 窗格中设置 TitleWindow 组件 id 属性为 winMain、title 属性为“我的第一个 Flex 程序”。切换到 Source 视图会看到生成的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:TitleWindow x="38" y="39" width="250" height="200" layout="absolute"
    id="winMain" title="我的第一个 Flex 程序">
    </mx:TitleWindow>
</mx:Application>
```

（8）使用上步的方法在 Controls 节点下拖曳一个 Label 组件到 TitleWindow 组件上。再设置 text 属性为 Welcome to Flex world!、id 属性为 lblStr、fontSize 属性为 16、color 属性为 FF001E。

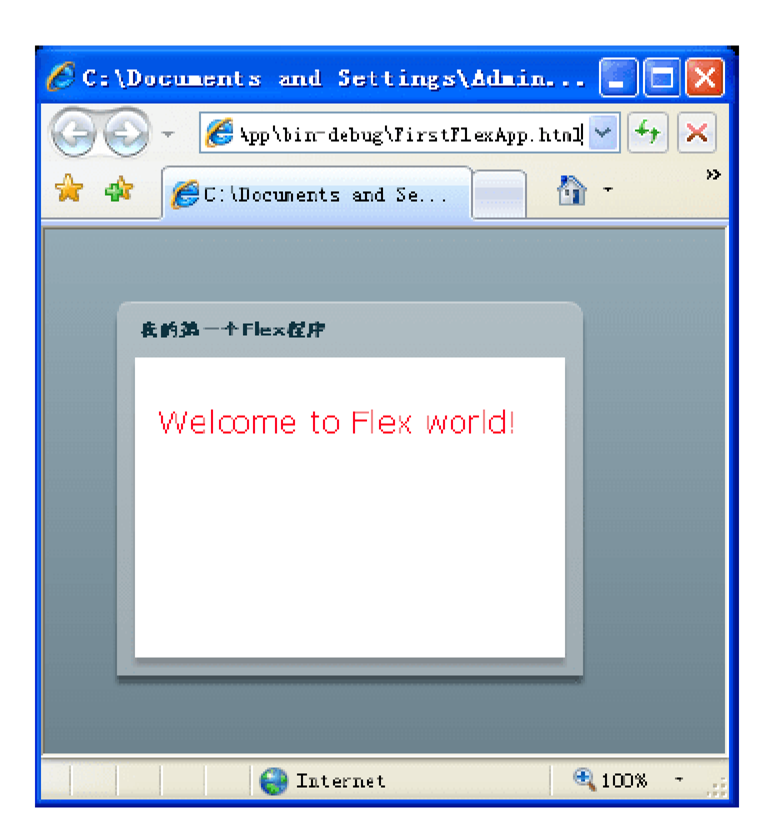
（9）在 Source 视图中为 Label 组件添加一个 click 属性，设置值为 ShowMsg('Flex')。这样定义一个单击事件，触发后执行 ShowMsg('Flex')。

（10）本实例采用级联方式调用 ActionScript 代码。接下来编写 ActionScript 代码，插入 <mx:Script>标签及触发后的实现代码。如下所示为最终的 MXML 文件内容。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:TitleWindow x="38" y="39" width="250" height="200" layout="absolute"
    id="winMain" title="我的第一个 Flex 程序">
        <mx:Label x="10" y="23" text="Welcome to Flex world!"
        id="lblStr" fontSize="16" color="#FF001E"
         click="ShowMsg('Flex')" width="210" height="25"/>
    </mx:TitleWindow>
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
```

```
                    private function ShowMsg(msgstr:String):void{
                        Alert.show("Hello "+msgstr);
                    }
            ]]>
        </mx:Script>
    </mx:Application>
```

（11）保存 FirstFlexApp.mxml 文件。在工具栏中单击 Debug 按钮对创建的 MXMLApplication 应用程序进行调试。此时 Flex Builder 3 会自动完成编译、部署到输出目录 bin-debug、创建相应的 HTML 文件并打开等操作。图 1-16 所示即为运行效果，可以单击 Welcome to Flex world!文字查看弹出的对话框。



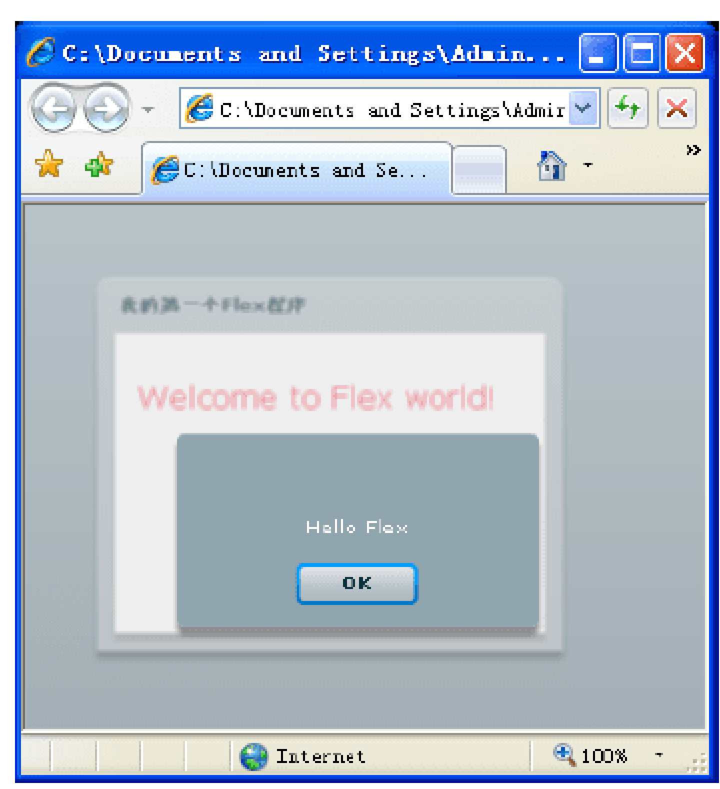
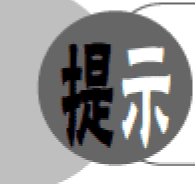


图 1-16　第一个程序运行效果



在本节创建第一个 Flex 程序的过程中会涉及到很多 Flex Builder 3 的操作。有关它的具体使用，将在下一章中详细介绍。

第2章 熟悉开发环境 Flex Builder 3



Flex Builder 3 是基于 Eclipse 的集成开发环境，用于开发能将桌面应用程序的丰富性和 Adobe 业务平台的跨平台性相结合的 RIA。Flex Builder 3 使开发人员能够快速构建能与 XML、Web 服务或者 Flex Data Services 集成的丰富的用户端逻辑。使用精准的设计和排版工具，用户界面设计人员能够创建更丰富多彩、可自定义外观的应用程序界面。本章将详细介绍 Flex Builder 3 的开发环境。

学习目标 Objective

- 熟悉 Flex Builder 3 的工作区
- 编译与运行 Flex 程序
- 使用断点
- 熟悉程序调试方法
- 监视变量
- 了解 Flex 3.0 项目及创建方法
- 了解 Flex Builder 3 的常用快捷键
- 学会使用 Flex Help

2.1 熟悉 Flex Builder 3 的工作区

Flex Builder 3 虽然功能非常强大，但是使用起来却并不难。本节将简要介绍 Flex Builder 3 的工作环境。

2.1.1 Editors

Editors，即编辑器，位于 Flex Builder 3 工作区中央的大块灰色区域是 Editor 窗格（Editor Pane），这是编写代码和修改代码，以及进行设计工作的地方。

在 Editor 窗格中，可以看到两种视图编辑模式：Source 视图和 Design 视图。下面将分别介绍这两种模式。

1．在 Source 视图下工作

MXML 代码存放在一个 MXML Application 文件内，大多数的 Flex 应用程序都只有或者说只需要一个 MXML Application 文件。当运行应用程序的时候，该文件（与 Flash 中的.fla 文件相似）就会编译生成一个 SWF 文件。

首先来看看 Flex Builder 3 在创建一个 MXML Application 文件时默认放置的代码，如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">

</mx:Application>
```

MXML Application 文件以一个 xml 声明开头，这是因为 MXML Application 文件必须遵守 XML 的规则，这些规则大致总结如下。

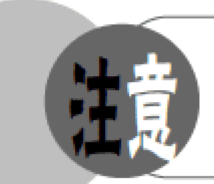
每个起始标签必须有一个对应的结尾标签。

标签区分大小写。

每个标签必须含有属性。

标签必须有严格的层级关系。

在一个 MXML Application 文件中，<mx:Application></mx:Application>标签对是根标签对，其他所有的标签必须位于根标签对之中。构建 Flex 应用程序的时候，实际是在构建一个多容器的配置，这些容器可以容纳其他的容器或者视觉元素，比如按钮、文本框、单选按钮、下拉列表框等。<mx:Application>标签除了被看作根标签外，还是一个主容器。



MXML Application 文件的标签都以标识符 mx:开头，这由 Flex 命名空间的特性决定。

下面，在 Source 视图下，设计一个简单的 Label 组件，显示一段文本 Flex is interesting，具体步骤如下所示。

（1）将光标放在两个 Application 标签之间。

（2）输入一个左尖括号<，这时 Flex Builder 3 马上就会给出一个可用标签列表，如图 2-1 所示。

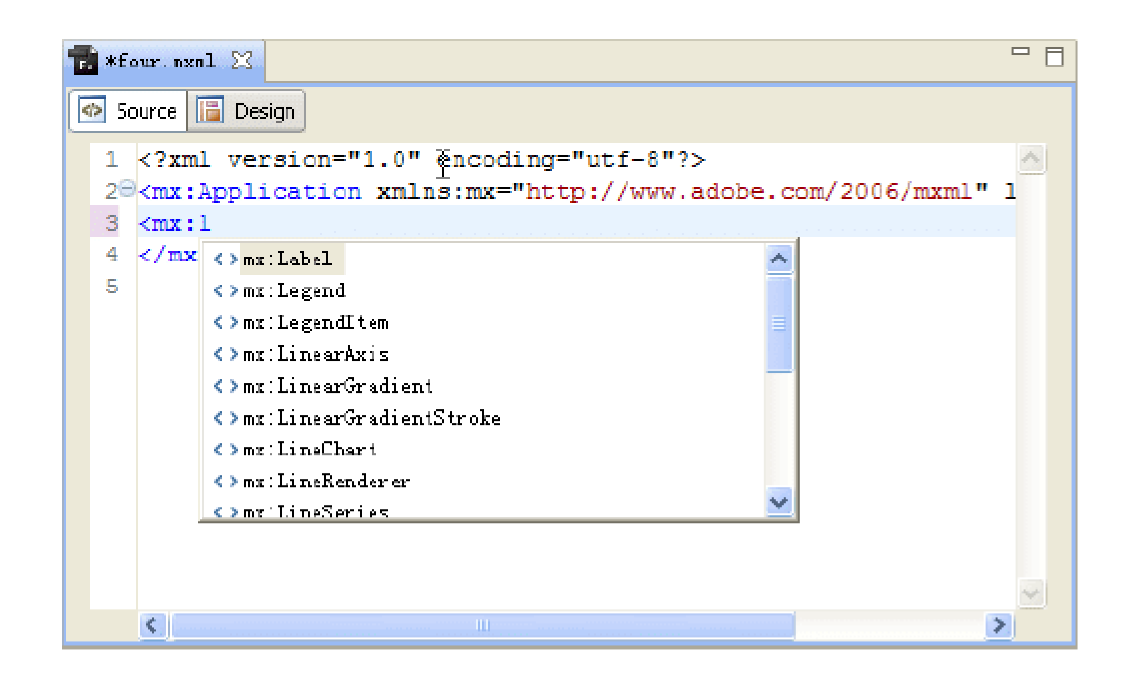
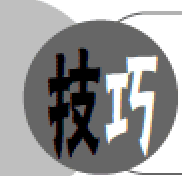


图 2-1　智能提示

（3）找到 mx:Label 或者直接输入 mx:Label，并且设置 text 属性为 Flex is interesting，具体代码如下所示：

```
<mx:Label text="Flex is interesting" />
```



如果标签内含有另外一个标签，就可以只用一个右尖括号>关闭标签；如果标签内没有其他标签，就可以使用/>来关闭标签。

（4）保存文件，单击 Run 按钮。具体效果如图 2-2 所示。

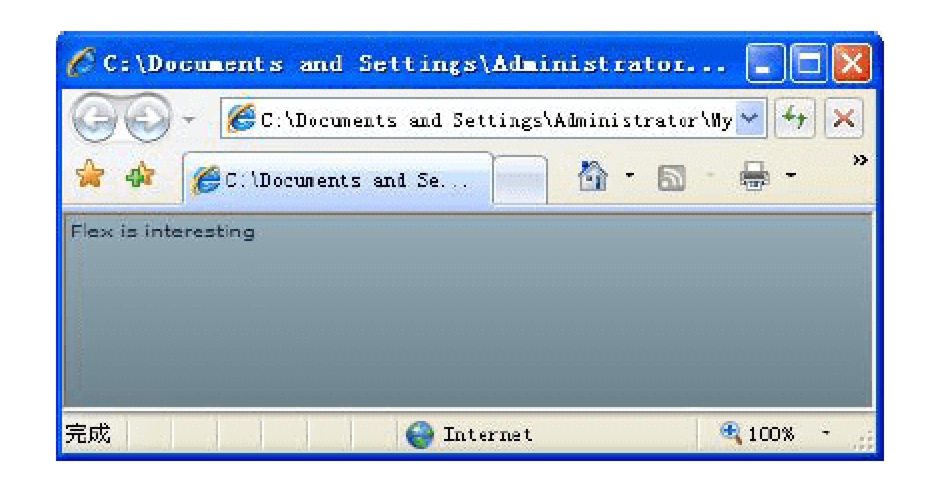


图 2-2　Flex 输出结果

2. 在 Design 视图下工作

单击 Design 按钮，就可以转到 Design 视图。在这里拥有一个以“所见即所得”方式设计网页的环境，如图 2-3 所示。

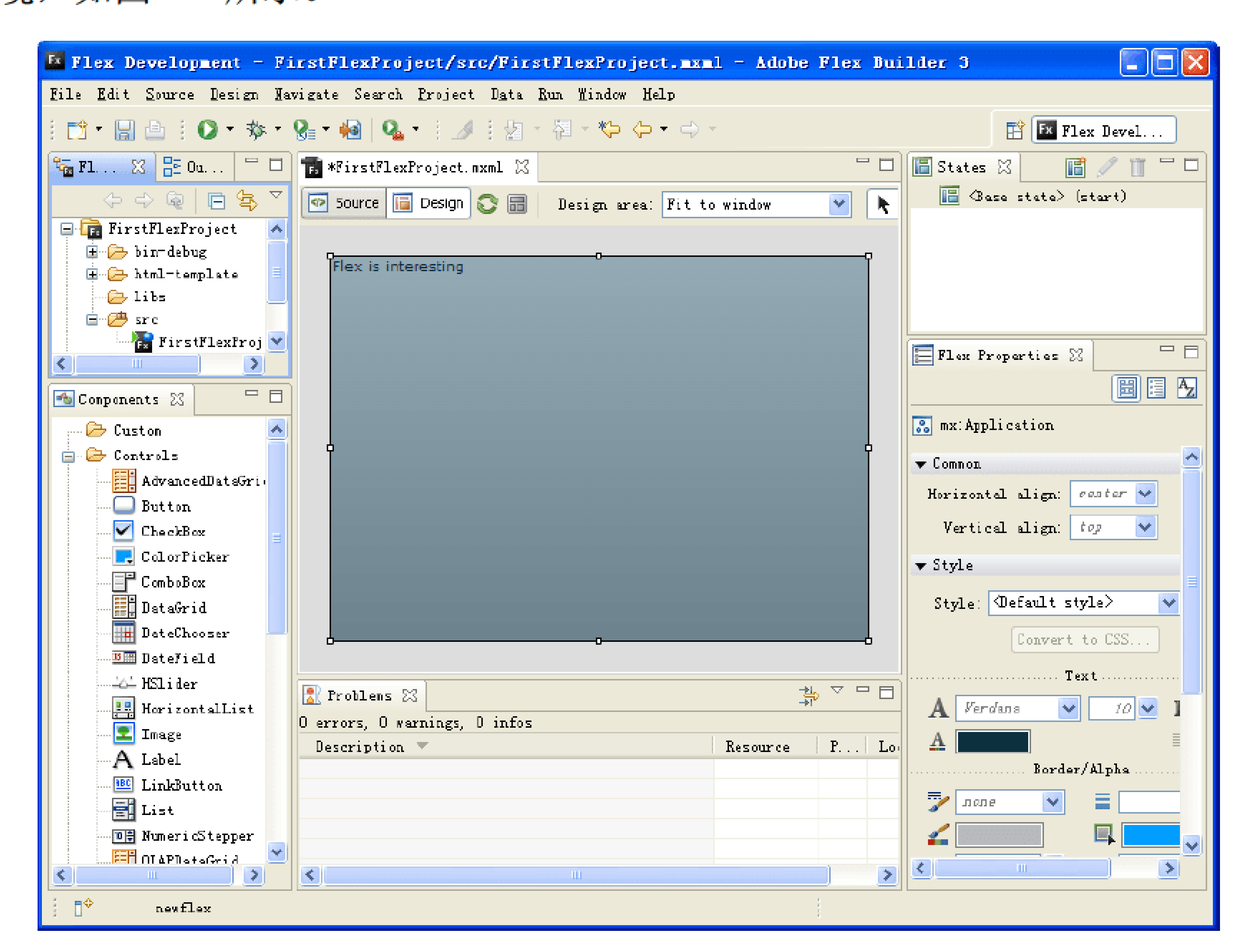


图 2-3　Design 视图

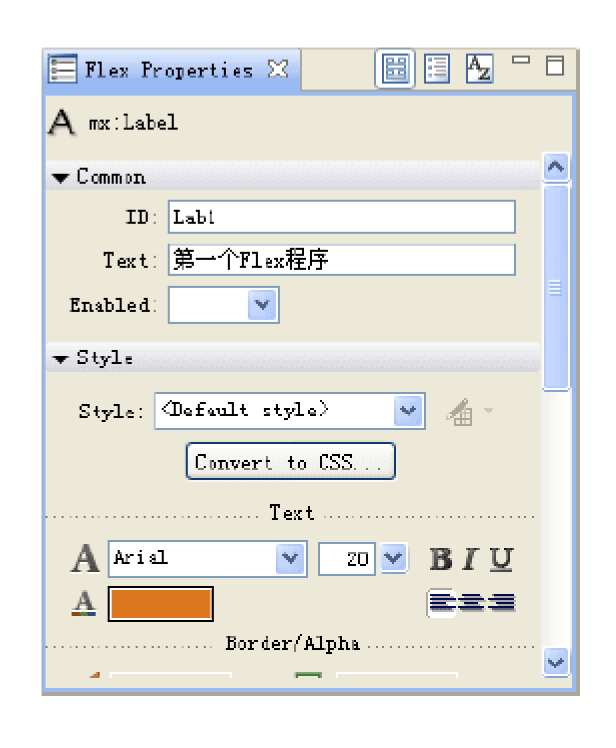
左下方的窗格，即 Components 窗格，显示的是所有可用组件的清单。右下方的属性窗格，即 Flex Properties 窗格，显示当前组件的所有属性。

接下来在 Design 视图中创建一个简单的页面，具体代码如下所示。

（1）从 Components 窗格中拖曳进来一个 Label 控件。

（2）设置 ID 属性为 Lab1，Text 属性为“第一个 Flex 程序”，Color 属性为“#DB771F”，FontSize 属性为 20，如图 2-4 所示。

（3）保存文件，单击 Run 按钮。具体效果如图 2-5 所示。



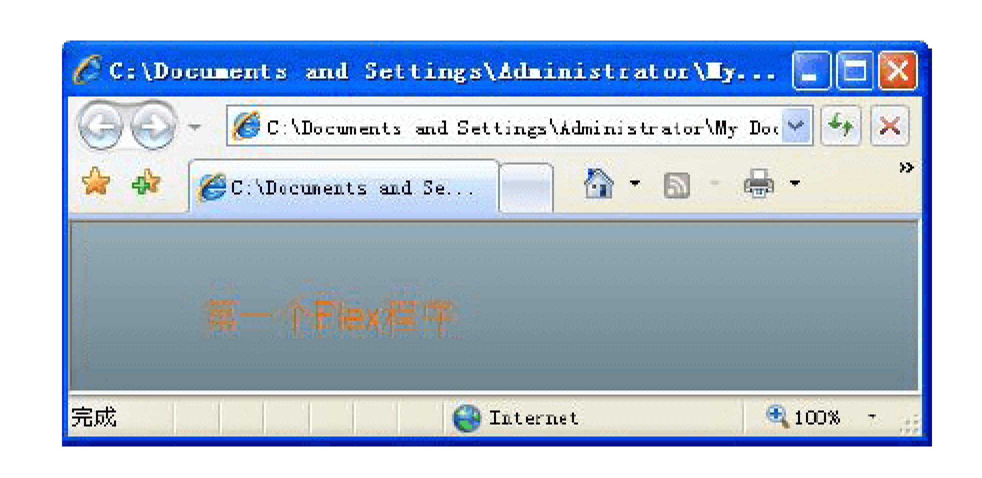



图 2-4　设置控件属性　　　　图 2-5　运行程序

在 Flex Properties 窗格顶部右边有几个按钮，其中第二个按钮叫作 Category View，单击这个按钮，Flex Properties 窗格会按照选项的类型显示出更多的分组选项，如图 2-6 所示。

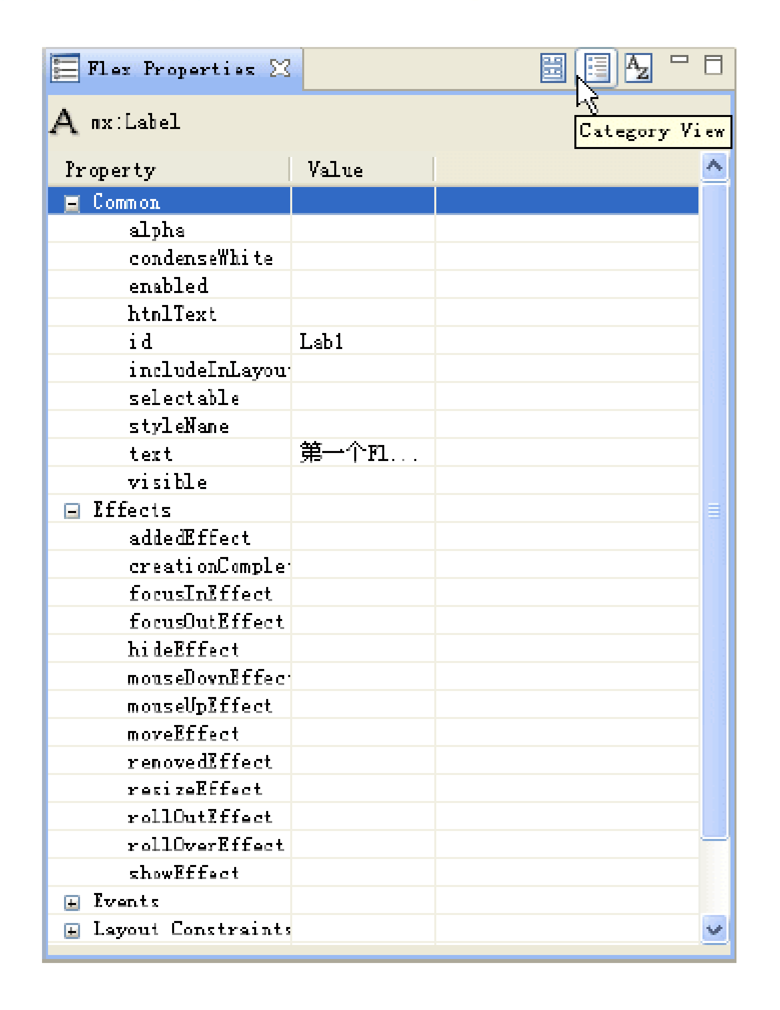


图 2-6　分类显示属性

在分类中，有一个类别是 Effects。在该类别中可以为控件添加动态效果。例如，设置 mouseDownEffect 属性的值为 WipeRight，在页面中单击该 Label 控件，就可以看到从左往右出现的动态效果。

2.1.2　其他窗格

上一节已经介绍了除了 Editors 编辑器外的一些窗格，比如 Compotents 窗格、Flex Properties

窗格等。单击顶部菜单栏中的 Window 命令，就列出了所有可用的窗格的列表，如图 2-7 所示。

1. Flex Navigator 窗格

Flex Navigator 窗格一般位于 Flex Builder 3 工作区的左上角，在该窗格中，显示当前项目的所用文件夹和文件。用户可以对项目的中的文件夹或者文件进行各种操作，比如添加、删除、重命名等，如图 2-8 所示。

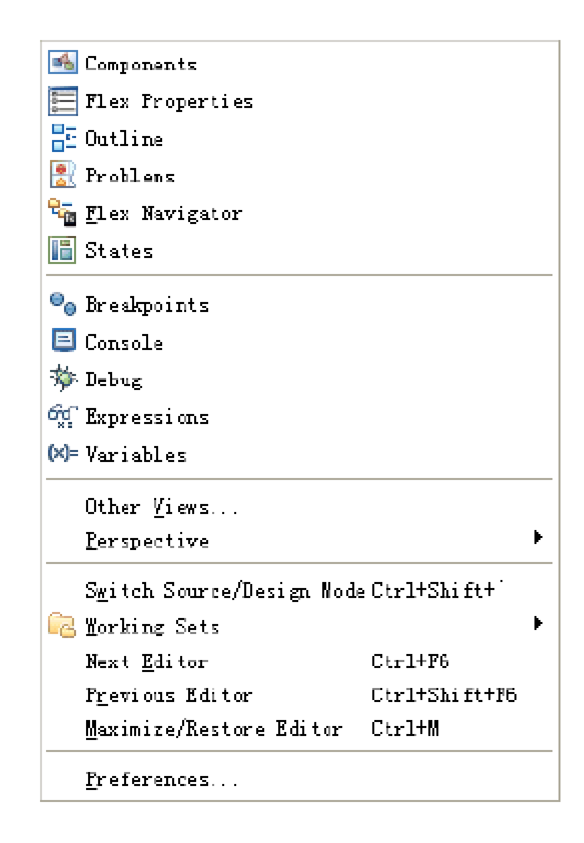




图 2-7 Window 命令菜单

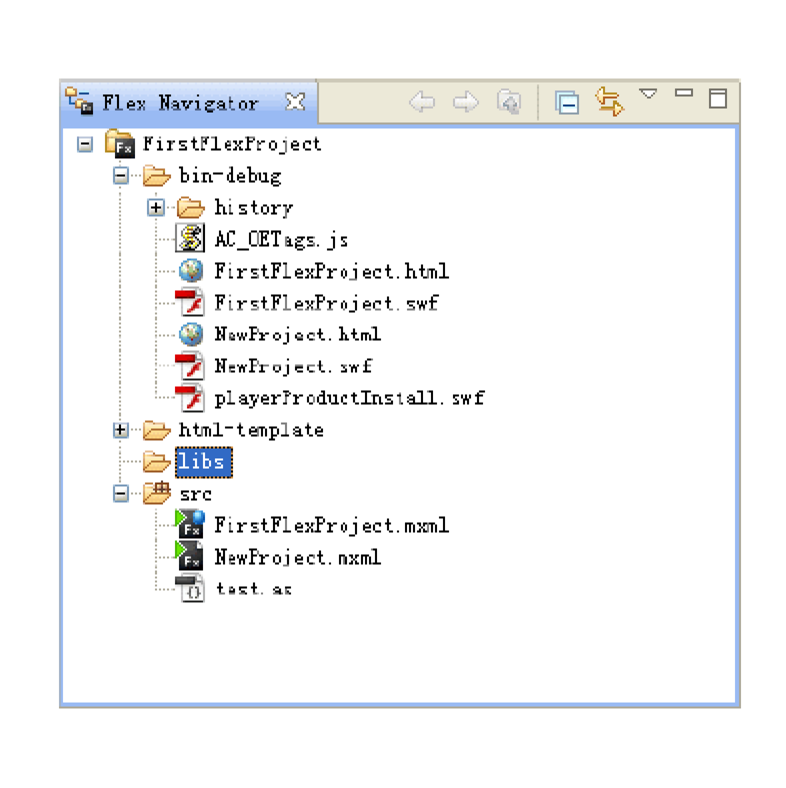


图 2-8 Flex Navigator 窗格

2. Components 和 Flex Properties 窗格

在 Components 窗格中，存放 Flex Builder 3 的所有可用控件，比如 Label 标签控件、Button 按钮控件、Image 图像控件等。Flex Properties 窗格则显示当前页面上每一个标记或者元素的属性集合。这两个窗格只在设计视图中有效，在代码视图中将没有任何内容。在前面介绍设计视图时，已经介绍过这两个窗格的内容，这里就不再重复。

3. Outline 窗格

Outline 窗格中显示当前页面所用到的所有元素。将所有元素按照各个元素的包容关系，以树状菜单形式显示出来，如图 2-9 所示。

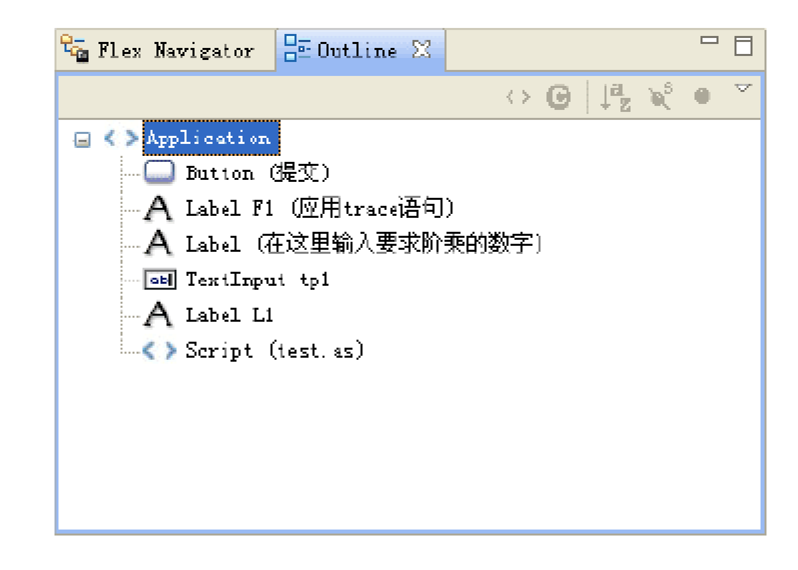


图 2-9 Outline 窗格

4. Problems 窗格

在代码模块，Flex Builder 3 会自动对代码进行检查，当发现用户设计的代码出现语法上

或者逻辑上的错误时，就会在 Problems 窗格中输出相应的错误或者警告信息，并且给出了该错误或者警告的具体描述和在程序文件中的具体位置，如图 2-10 所示。双击该错误信息，就可以直接转到程序中出现错误的语句行。通过 Problems 窗格的提示，避免了一些因为用户疏忽而造成的错误，也大大减少了程序调试的工作量。

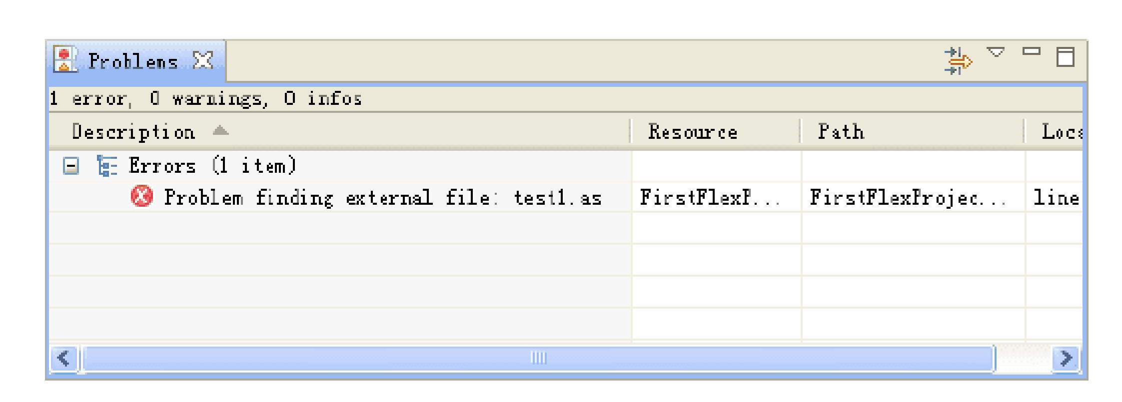


图 2-10　Problems 窗格

5. States 窗格

在 Flex Builder 3 中可以使用视图状态和变换来创建更为丰富、更具互动性的用户体验。例如，可以使用视图状态去创建用户界面，根据用户所执行的任务来改变它的外观。

视图状态（View States）为一个 MXML 程序或者组件定义的布局命名。可以为一个程序或者组件定义几种视图状态，并且根据用户的行为在之间进行切换。视图状态允许动态地改变用户界面，以便对用户的活动或者增加的内容做出响应。

在 States 窗格中，用户可以新建状态，通过在页面中调用不同的状态，改变页面的外观和内容。

在 States 窗格中右击，在弹出的命令菜单中选择 New state 或者直接单击就可以新建状态。这里创建了一个名称为 Register 的新状态，如图 2-11 所示。

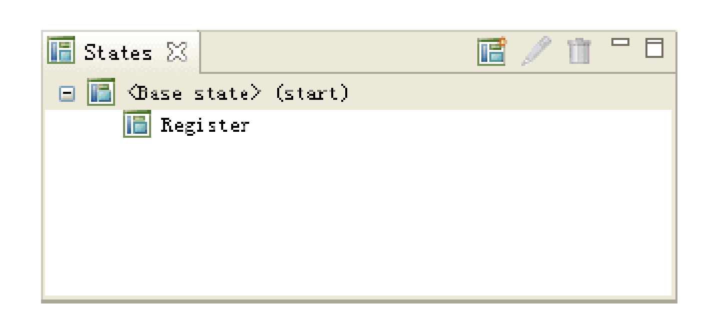


图 2-11　States 窗格

状态创建完成后，在代码视图中对该状态的内容进行编辑，这里设置新状态显示【注册】界面，在原来【登录】界面的基础上增加“确认密码：”项，并改变控件的相应属性，设置超链接文本的 click 事件返回初始状态，当用户单击“返回登录”超链接文本时，返回【登录】界面。具体代码如代码 2.1 所示。

代码 2.1　登录界面代码

```
<mx:states>
    <mx:State name="Register">
        <mx:AddChild relativeTo="{loginForm}" position="lastChild">
            <mx:target>
                <mx:FormItem id="confirm" label="确认密码:">
                    <mx:TextInput/>
```

```
                </mx:FormItem>
            </mx:target>
        </mx:AddChild>
        <mx:SetProperty target="{loginPanel}" name="title" value="注册"/>
        <mx:SetProperty target="{loginButton}" name="label" value="注册"/>
        <mx:SetStyle target="{loginButton}"
            name="color" value="blue"/>
        <mx:RemoveChild target="{registerLink}"/>
        <mx:AddChild relativeTo="{spacer1}" position="before">
            <mx:target>
                <mx:LinkButton id="loginLink" label="返回登录" click=
                "currentState=''"/>
            </mx:target>
        </mx:AddChild>
    </mx:State>
</mx:states>
```

在页面中运行，具体的效果如图 2-12、图 2-13 所示。

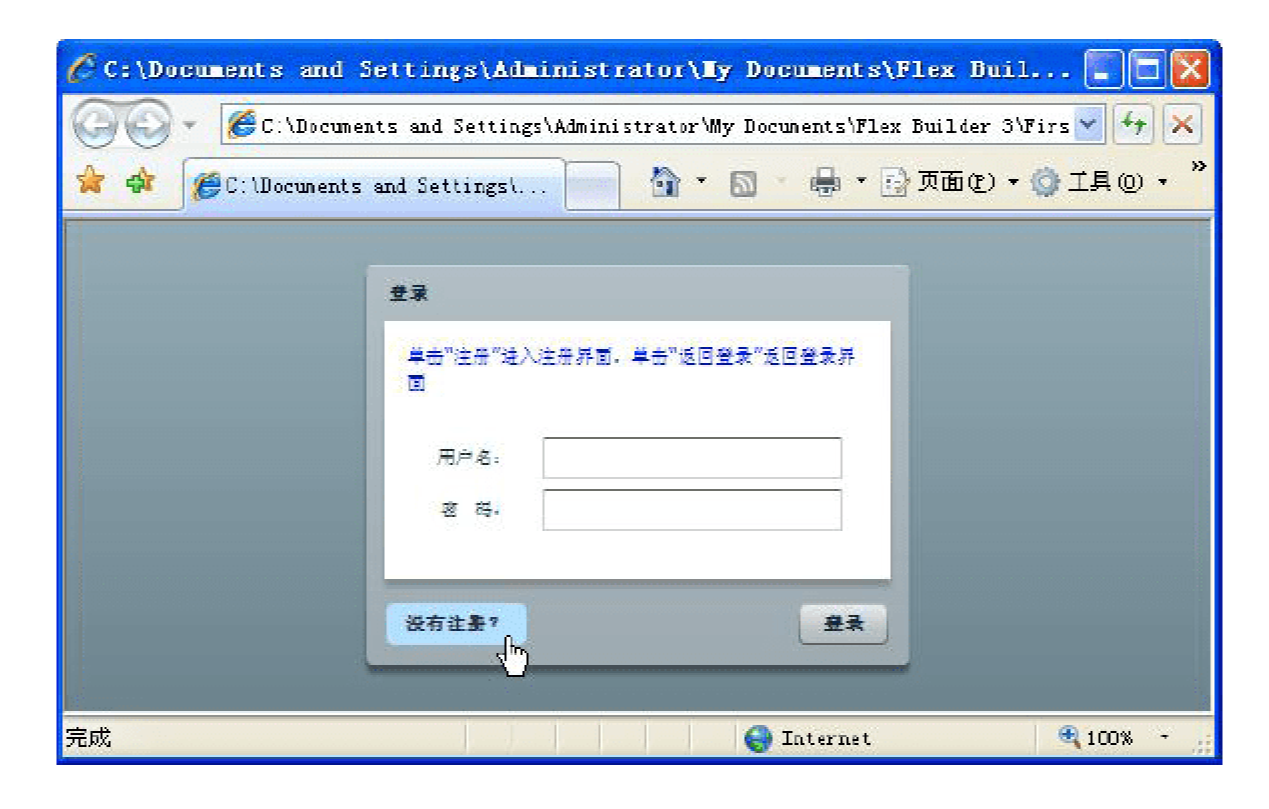


图 2-12 【登录】界面

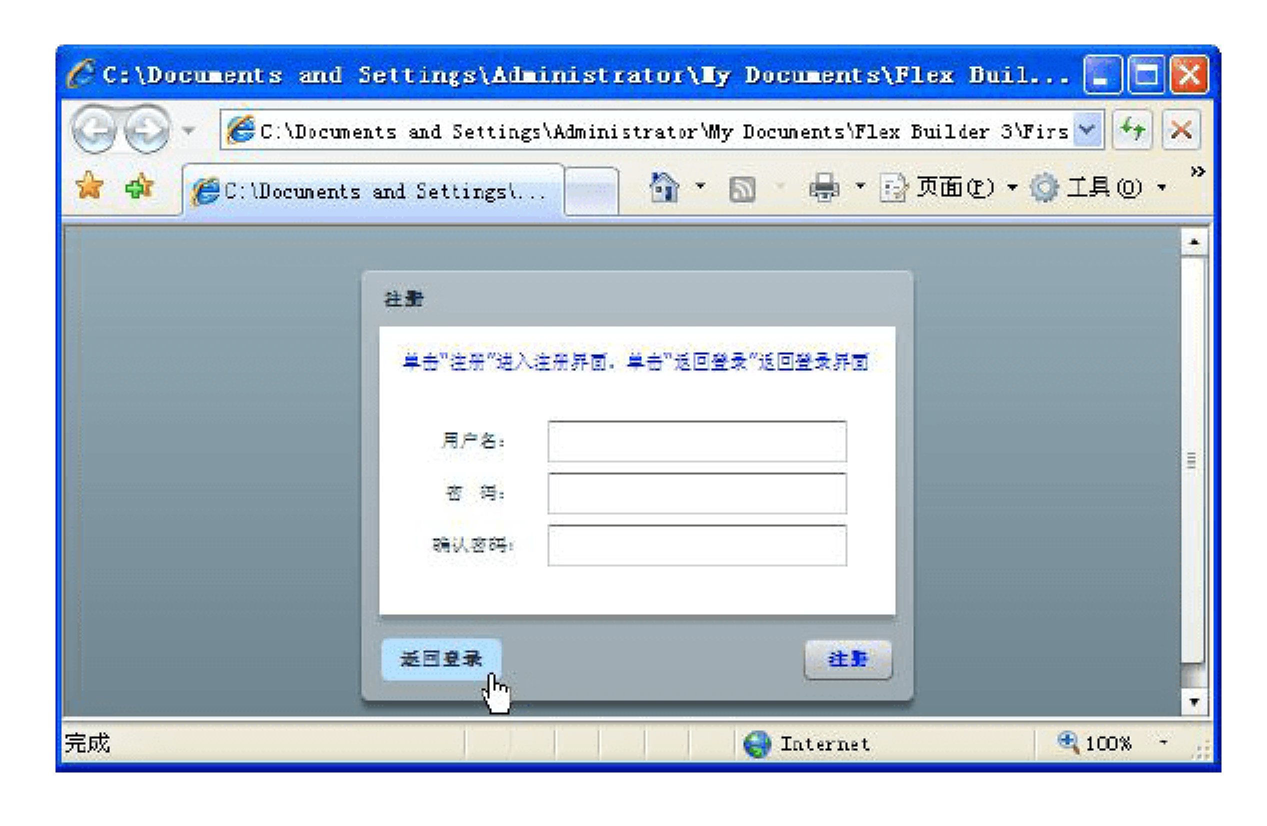


图 2-13 【注册】界面

6．Debug 和 Console 窗格

页面和代码设计完成后，选择命令，对程序进行调试。程序执行结束后，返回 Flex Builder 3 界面，就可以看到 Debug 窗格和 Console 窗格。

Debug 窗格显示当前文件的执行进度和状态，如图 2-14 所示。

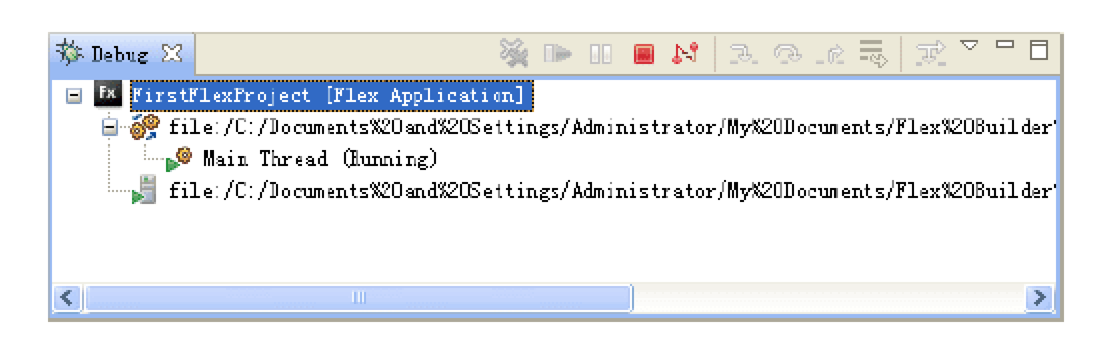


图 2-14　Debug 窗格

如果在程序中使用了调试语句 trace，那么在 Console 窗格中就会出现生成的 SWF 文件及使用 trace()语句查看的变量的值，如图 2-15 所示。

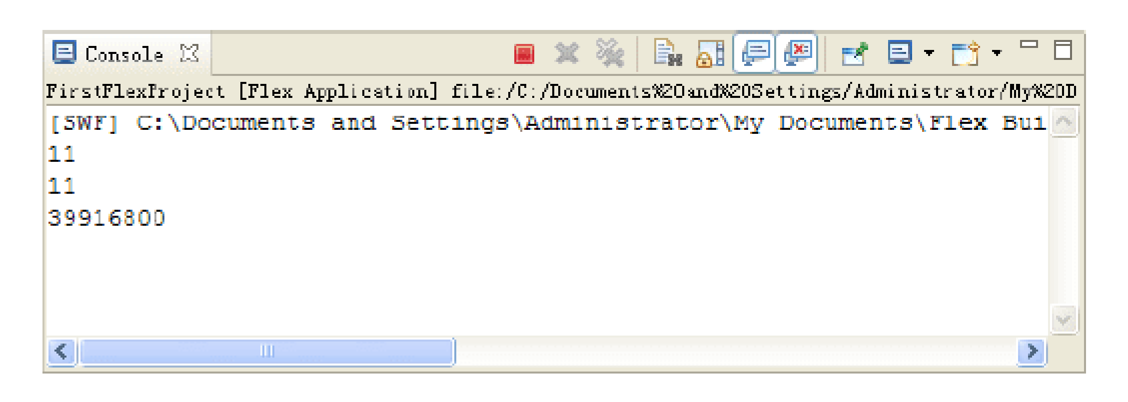


图 2-15　Console 窗格

7．Variables、Expressions 和 Breakpoints 窗格

在调试模式下，除了 Debug 窗格和 Console 窗格外，还有 3 个窗格，分别是 Variables、Expressions 和 Breakpoints。

Breakpoints 窗格显示了当前项目所有的文件中断点的位置，如图 2-16 所示。双击该断点，就可以直接转到设置断点的位置。

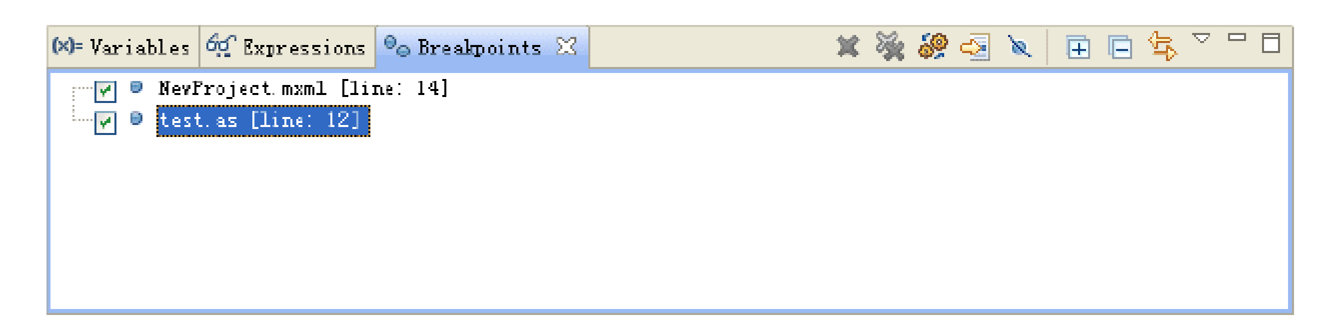


图 2-16　Breakpoints 窗格

Variables 窗格监视正在调试的程序文件中的所有变量的值，并将这些变量的值以列表的形式显示出来，如图 2-17 所示。程序执行完成后，Variables 窗格将释放所监视的所有变量的值。

在 Expressions 窗格中，可以监视一个或者多个特定的表达式，在调试程序时，显示所有的表达式及值。

图 2-17　Variables 窗格

2.2　编译与运行 Flex 3.0 程序

程序编写完成，就可以编译、运行这个项目了。首先确认选中了当前项目，或者选中了当前项目中的程序文件，单击工具栏中的运行图标，开始编译。状态栏的进度条提示 Flex Builder 正在工作，如果程序有错误，将中止编译动作，并在 Problems 窗格中列出错误信息。如果没有问题，编译结束后 Flex Builder 自动打开一个新的浏览器窗口，来浏览 HTML 页面。这个 HTML 页面中嵌入了生成的 SWF 文件，如图 2-18 所示。

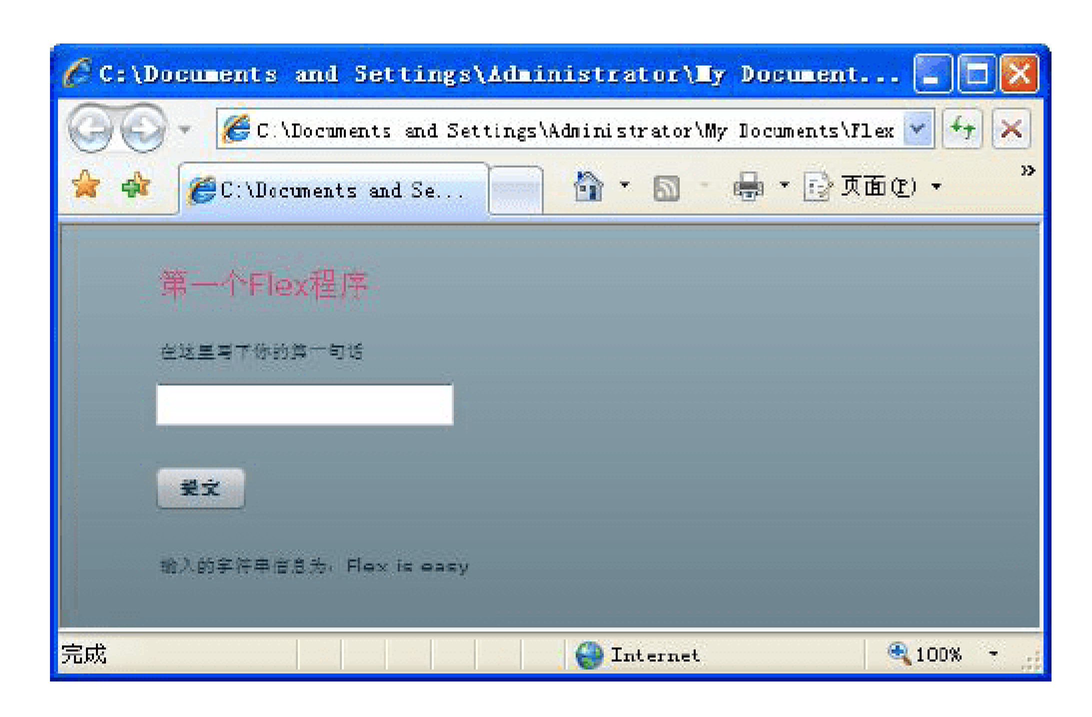


图 2-18　程序执行成功

在页面中单击【确定】按钮，Label 组件显示用户刚刚输入的字符串，说明程序成功运行。

2.3　调试 Flex 3.0 程序

并不是所有的程序都会成功执行，大多数时候，在程序编译的时候会出现这样那样的问题，这时就需要对程序进行调试。Flex Builder 中强大的调试工具可以帮助用户隔离代码中的错误，建立最佳的应用程序。Flex Builder 的 ActionScript 调试器利用与 Flex 环境的紧密集成，可以更快速地识别 ActionScript 错误。

2.3.1　添加断点

断点是一个信号，它通知调试器，在某个特定点上暂时将程序执行挂起。当执行在某个断点处挂起时，称程序处于中断模式。进入中断模式并不会终止或者结束程序的执行。执行

可以在任何时候继续。

断点模式可以看作一种超时状态，在该状态下，所有元素（例如，函数、变量和对象）都保留在内存中，但它们的移动和活动被挂起。开发者可以检查它们的位置和状态，以查看是否存在冲突或者 bug。可以在中断模式下对程序进行调整。例如，可以更改变量的值，可以移动执行点，这会改变执行恢复后将要执行的下一条语句。

断点提供了一种强大的功能，使能够在需要的时间和位置挂起执行。与逐句或者逐条指令地检查代码不同，可以让程序一直执行，直到遇到断点，然后开始调试。这大大地加快了调试过程。没有这个功能，调试大的程序几乎不可能。

许多编程语言都有用于挂起执行并使程序进入中断模式的语句或者构造。断点不同于这些语句，因为它不是必须添加到程序中的实际源代码。不必在源代码窗口中输入断点语句。只需通过调试器界面请求断点，由调试器加入断点。相对于调试构造，断点具有很多优点：不必改动程序源代码就可以删除或者更改断点。由于断点不是语句，当生成程序的发行版时，不会产生额外的代码。

在 Flex Builder 3 中的代码视图中，在需要设置断点的行号前双击，就可以设置断点，如图 2-19 所示。

```
FirstFlexProject.mxml   test.as
// ActionScript file
import mx.controls.Alert;
public function alert(str:String):void
{
    L1.text="输入的字符串信息为："
    L1.text+=tp1.text;
    trace(L1.text);
    Alert.show(L1.text,"你输入的信息是");
    tp1.text="";
}
```

图 2-19　设置断点

设置过断点后，单击命令，开始调试程序。当用户输入信息并单击按钮后，执行具体的功能代码。当执行到设置断点的位置时，会停止执行，在 Flex Builder 3 中显示已经生成的文本内容，如图 2-20 所示。

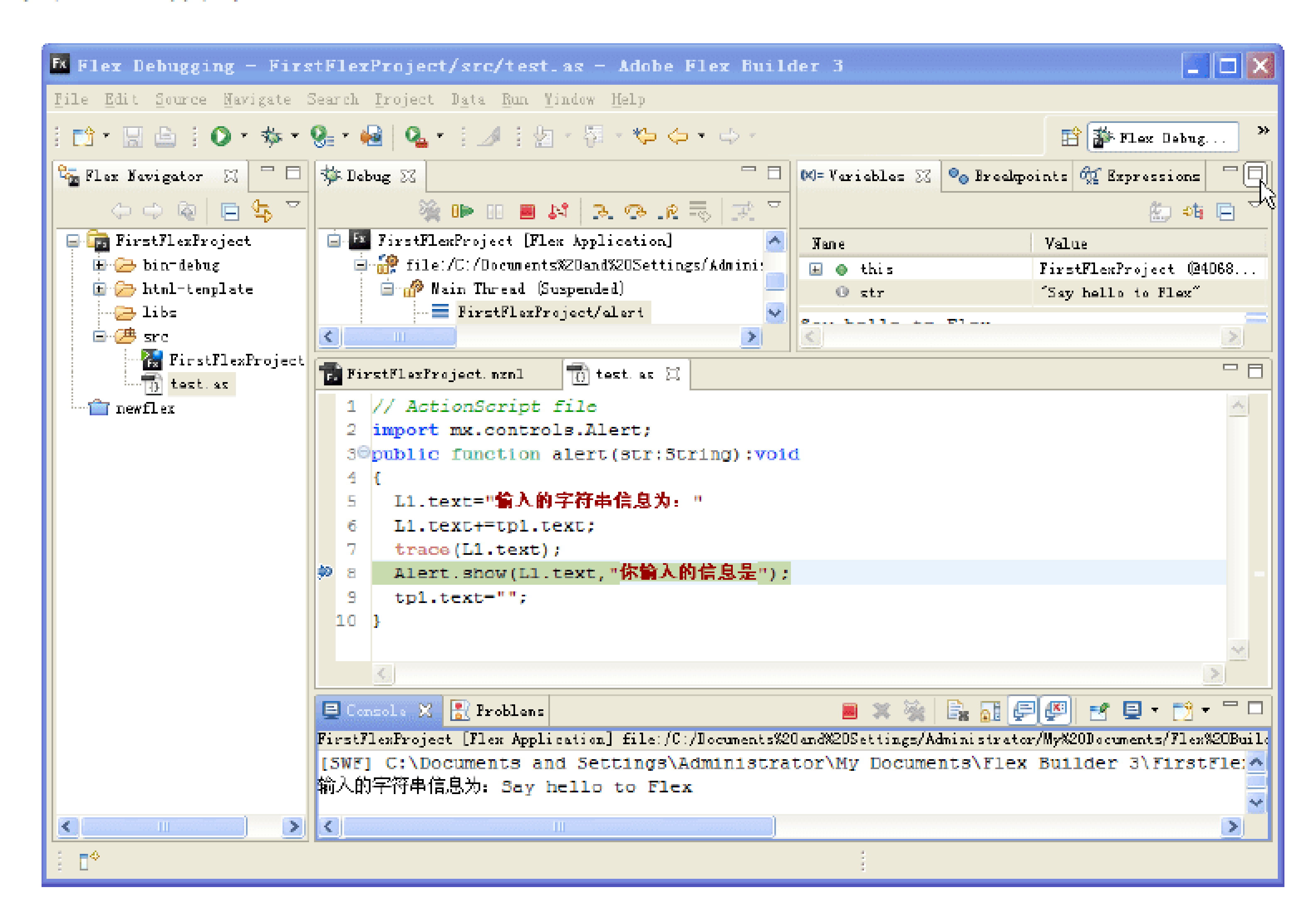


图 2-20　执行断点

确认断点前面的程序段没有错误后，单击【逐步】按钮执行下一句代码，单击【跨步】按钮返回执行上一句代码，单击【恢复】按钮继续执行程序，单击【终止】按钮终止执行程序。

2.3.2 调试程序

在调试程序的过程中，trace()语句非常重要。在调试文件时，使用此语句可在 Console 窗格中记录编程注释或者显示消息。使用 expression 参数可以检查是否存在某种条件，或者在 Console 窗格中显示值。trace()语句类似于 JavaScript 中的 alert()函数。

Trace()语句的语法如下：

```
trace(expression:Object)
```

expression 参数表示输出特定的变量的值或者检查是否存在某种条件。

当程序出现错误的时候，尽管会有错误的相应提示，但是有些错误还是很难找到。这时就可以在程序中使用 trace()语句进行调试，通过 Console 窗格中的内容，就可以定位出现错误的大概位置，方便用户对程序进行修改。

例如在代码 2.2 的程序中加入 trace()语句。

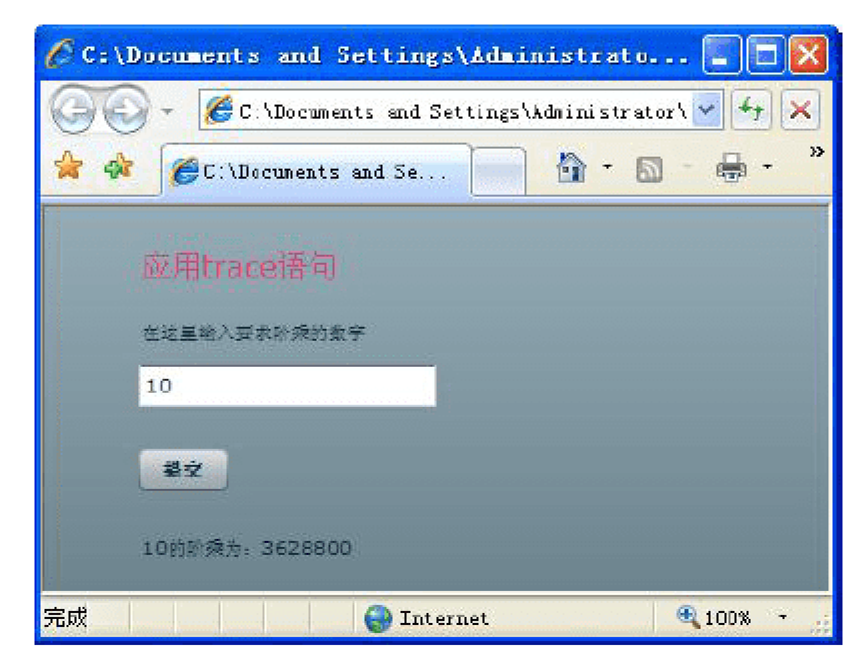
代码 2.2 使用 trace()语句

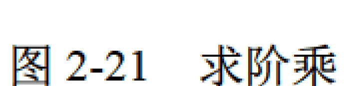
```
public function alert(str:String):void
{
  trace(tp1.text);
  var res:int=1;
  var num:int=int(str);
  trace(num);
  for(var i:int=1;i<=num;i++)
  {
     res*=i;
  }
  trace(res);
  L1.text=tp1.text+"的阶乘为: "+res;
}
```

在代码 2.2 这段程序中，创建了一个函数，根据用户输入的值求出阶乘。在函数中，使用了 3 个 trace 语句，分别获取用户输入的值、转换类型后的变量 num 和最终的结果 res。在调试应用程序的时候，就可以根据 Console 窗格中的变量输出结果，判断程序是否正确执行。如果出现错误，也可以根据输出结果，判断出程序执行到那里，从而快速地定位到错误的位置。

单击命令，开始调试，在文本框中输入一个值，单击【确定】按钮。该程序的执行结果如图 2-21 所示。

返回到 Flex Builder 3 中的 Console 窗格，这里列出了程序中定义的需要查看的变量值。如图 2-22 所示。




图 2-21　求阶乘

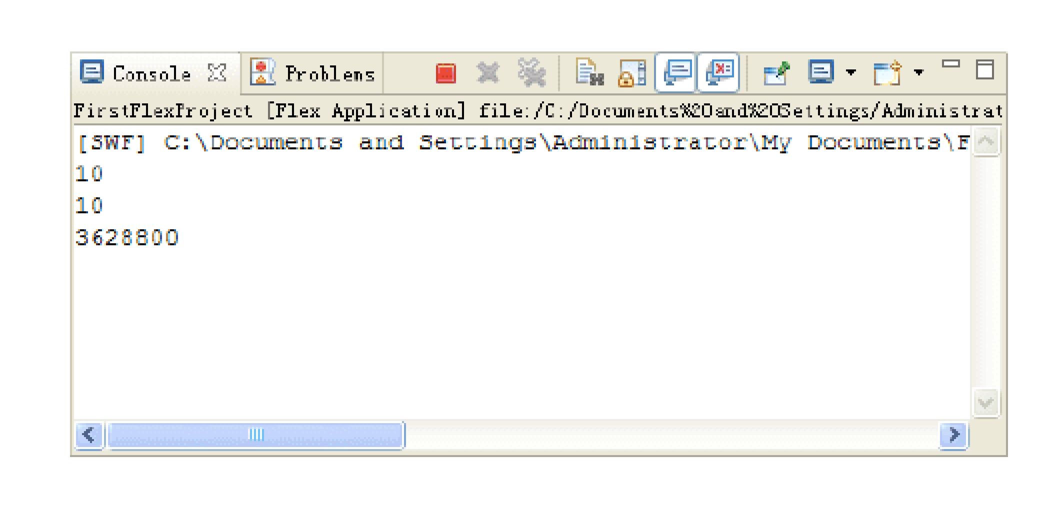


图 2-22　Console 窗格

2.3.3　监视变量

在 2.3.2 节中介绍了如何使用 trace()语句来调试程序，获取关键变量的值并输出。那么除了使用 trace()语句外，在 Flex Builder 3 中，在程序中设置断点，也可以完成监视变量的功能。

下面使用一个简单的实例，来演示如何使用断点在调试过程中监视变量。该实例的代码如代码 2.3 所示。

代码 2.3　监视变量

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Button label="求平方" click="alert(tp1.text)" x="50" y="130"/>
    <mx:Label x="50" y="21" text="监视变量" width="213" id="F1" color="#DA4375"
    height="30" fontSize="16"/>
    <mx:Label x="50" y="59" text="在这里输入要操作的数字" width="177"/>
    <mx:TextInput x="50" y="85" id="tp1"/>
    <mx:Label x="50" y="174" width="249" height="29" id="L1"/>
    <mx:Script>
        <![CDATA[
        public function alert(str:String):void
        {
             var num:int=int(str);
            var res:int=num*num;
            L1.text=tp1.text+"的平方为: "+res;
        }
         ]]>
    </mx:Script>
</mx:Application>
```

在函数中，设置了一个断点，在调试的时候，在 Variables 窗格中就会监视所有变量，并且显示所有变量的值，如图 2-23 所示。

单击【继续】按钮继续执行程序，该程序具体的执行结果如图 2-24 所示。

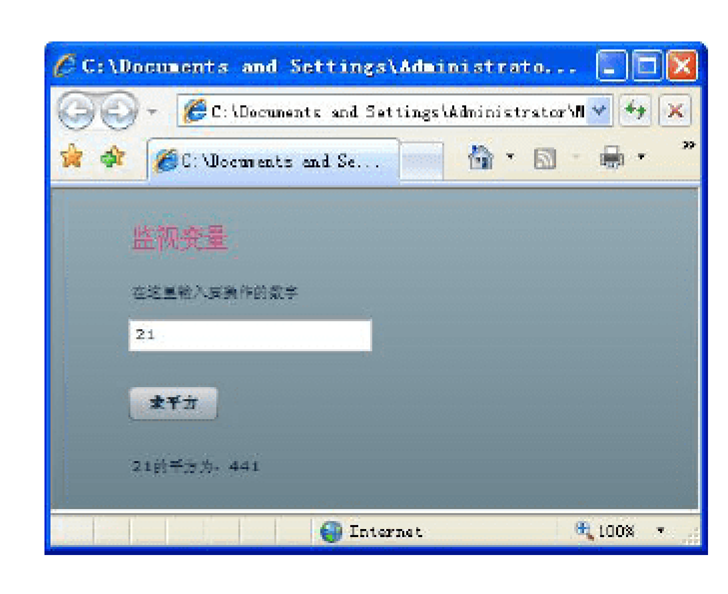




图 2-23　监视变量　　　　图 2-24　执行结果

2.4　Flex 3.0 项目概述

同 Flash 相比，Flex 加强了文档的管理功能，使开发者可以非常轻松地管理所有的文件。这继承了 Eclipse 的优良传统。一个程序所用到的所有资源都包含在一个集合中，也就是一个项目的文件夹中。

在 Flex Builder 3 中，在 File 菜单中可以创建 3 种项目，分别是 Flex Project、ActionScript Project 和 Flex Library Project。在本节中将分别介绍这 3 种项目的具体作用及创建方法。

2.4.1　Flex Project

Flex Project，基于 Flex Framework，是本书的主要讲解内容。下文中所涉及到的实例都是使用这种形式创建的程序项目。该类项目也可以分为两种类型：Web application 和 Desktop application。

1. Web application

Web application 类型的 Flex Project 项目编译时生成 HTML 文件，在浏览器中执行。下面演示一下创建 Web application 类型的 Flex Project 的具体流程。

（1）选择 File|New | Flex Project 命令，如图 2-25 所示。

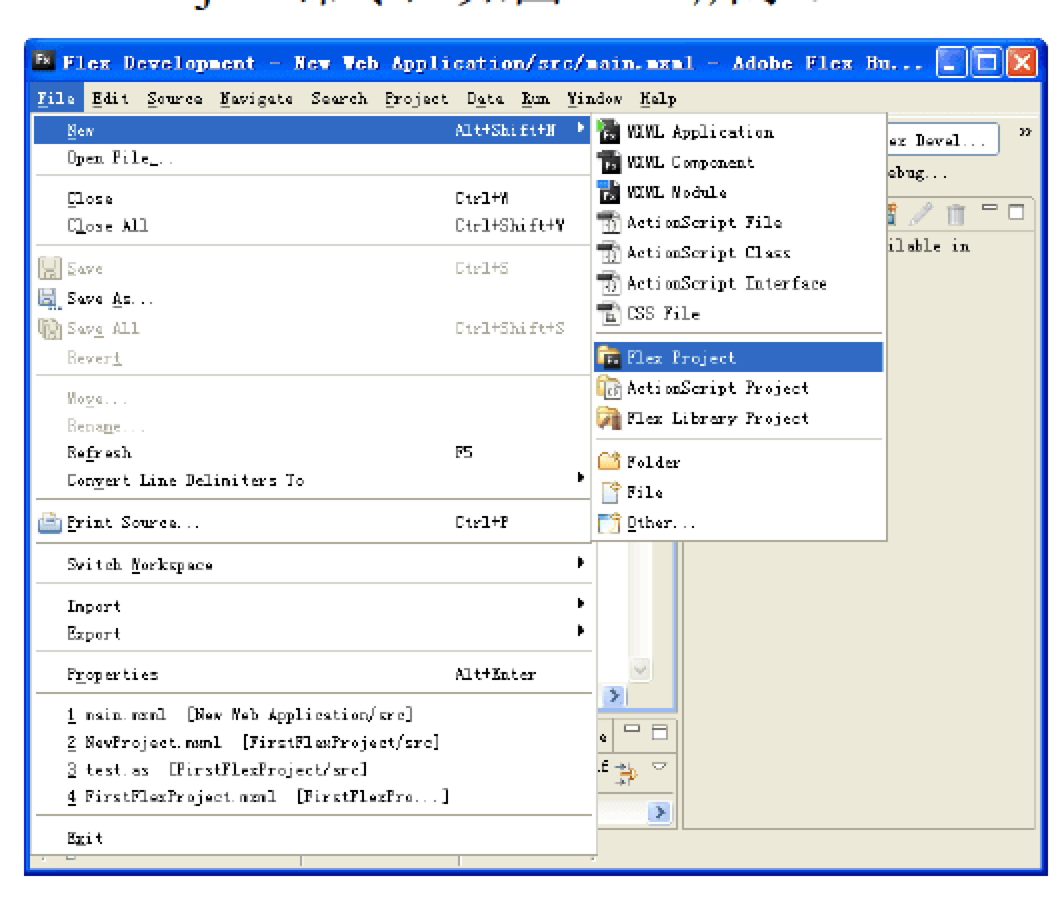


图 2-25　创建 Flex Project

（2）输入项目名称 New Web Application，选择该项目的存放路径，在 Appliction type 下面选择 Web application 选项，如图 2-26 所示。

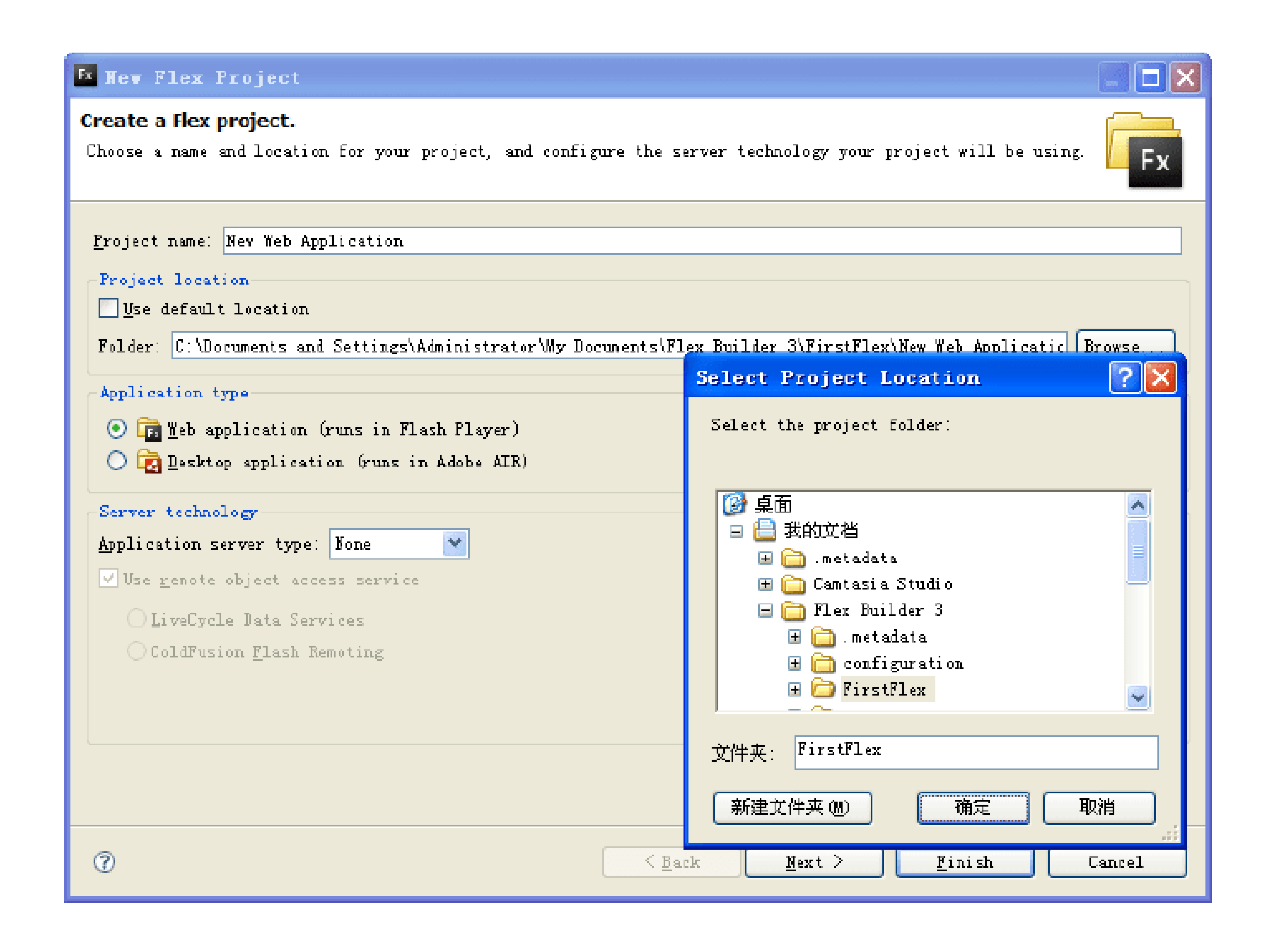


图 2-26　设置项目文件信息

（3）配置完成后单击 Next 按钮，进入 Configure Output 窗口，在该页面选择用于输出 SWF 和 HTML 文件的文件夹，如图 2-27 所示。

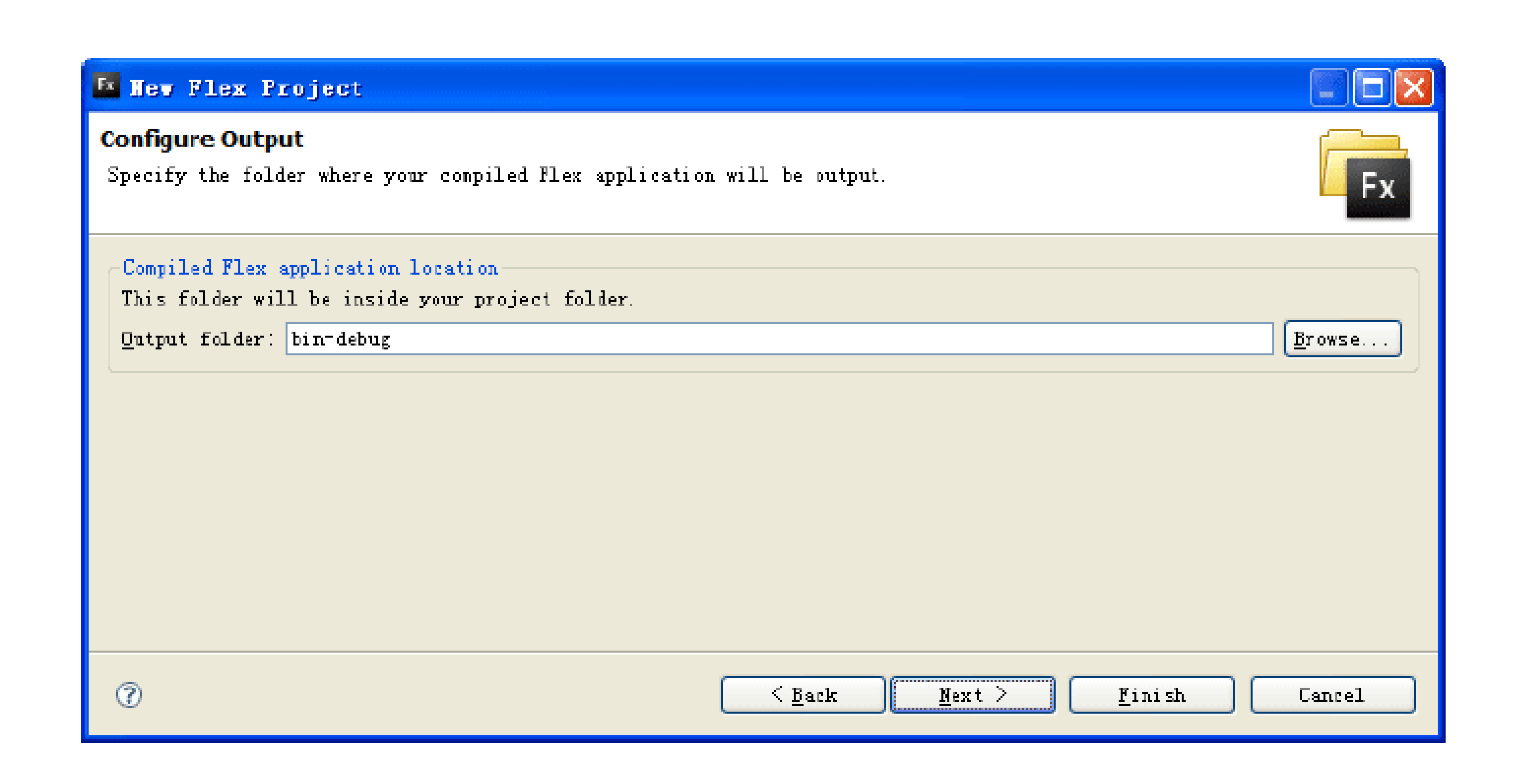


图 2-27　设置输出文件夹

（4）设置完成后，单击 Next 按钮，设置该项目存放程序文件的文件夹和主文件名称，如图 2-28 所示。

（5）选择 Library path 选项，设置 Flex 类库文件的存放位置，如图 2-29 所示。

（6）设置完成后，单击 Finish 按钮，完成项目的创建。在 Flex Navigator 窗格中，就可以看到该项目的所有文件夹和主文件，如图 2-30 所示。

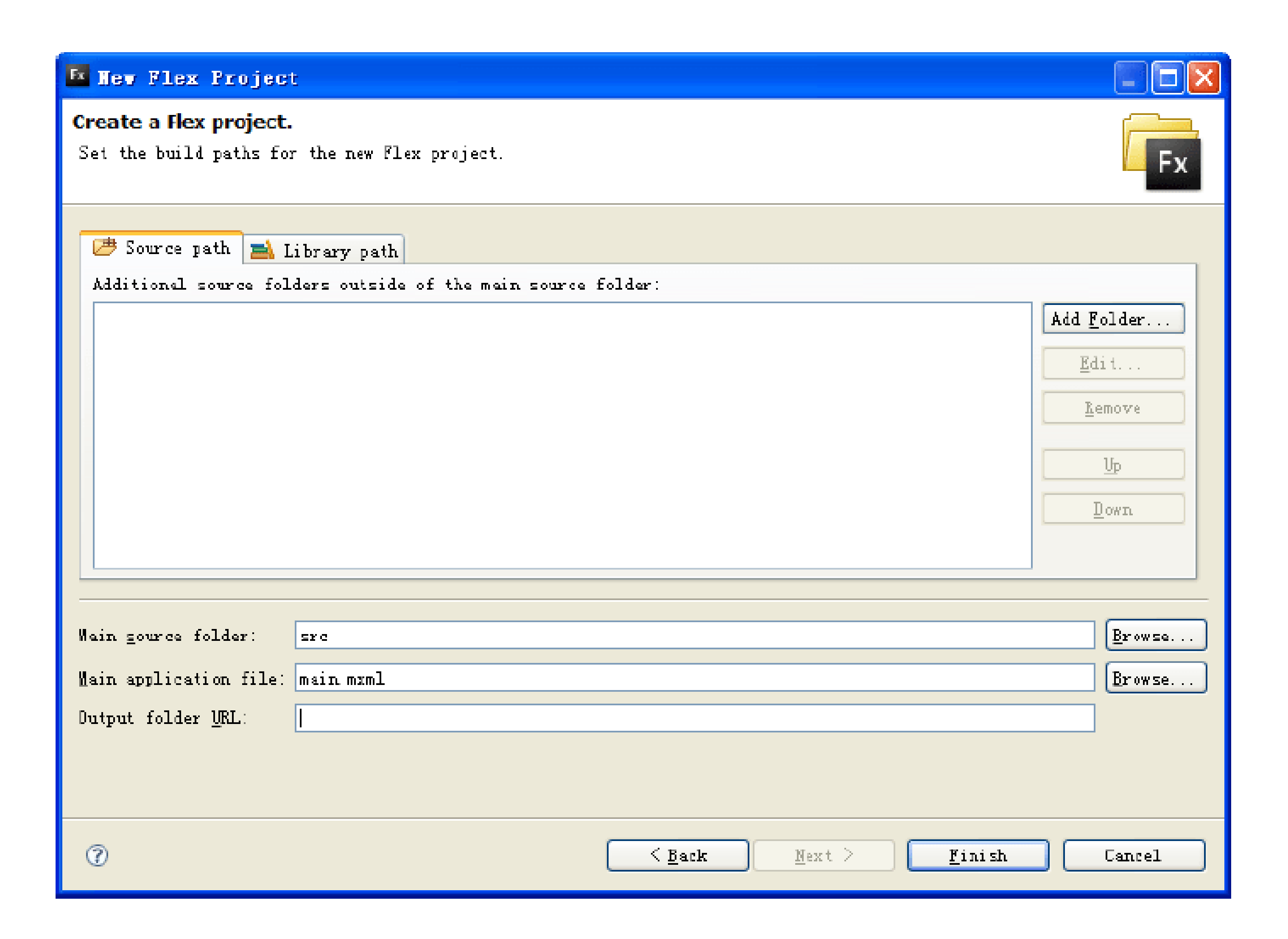


图 2-28　设置主文件夹和主文件

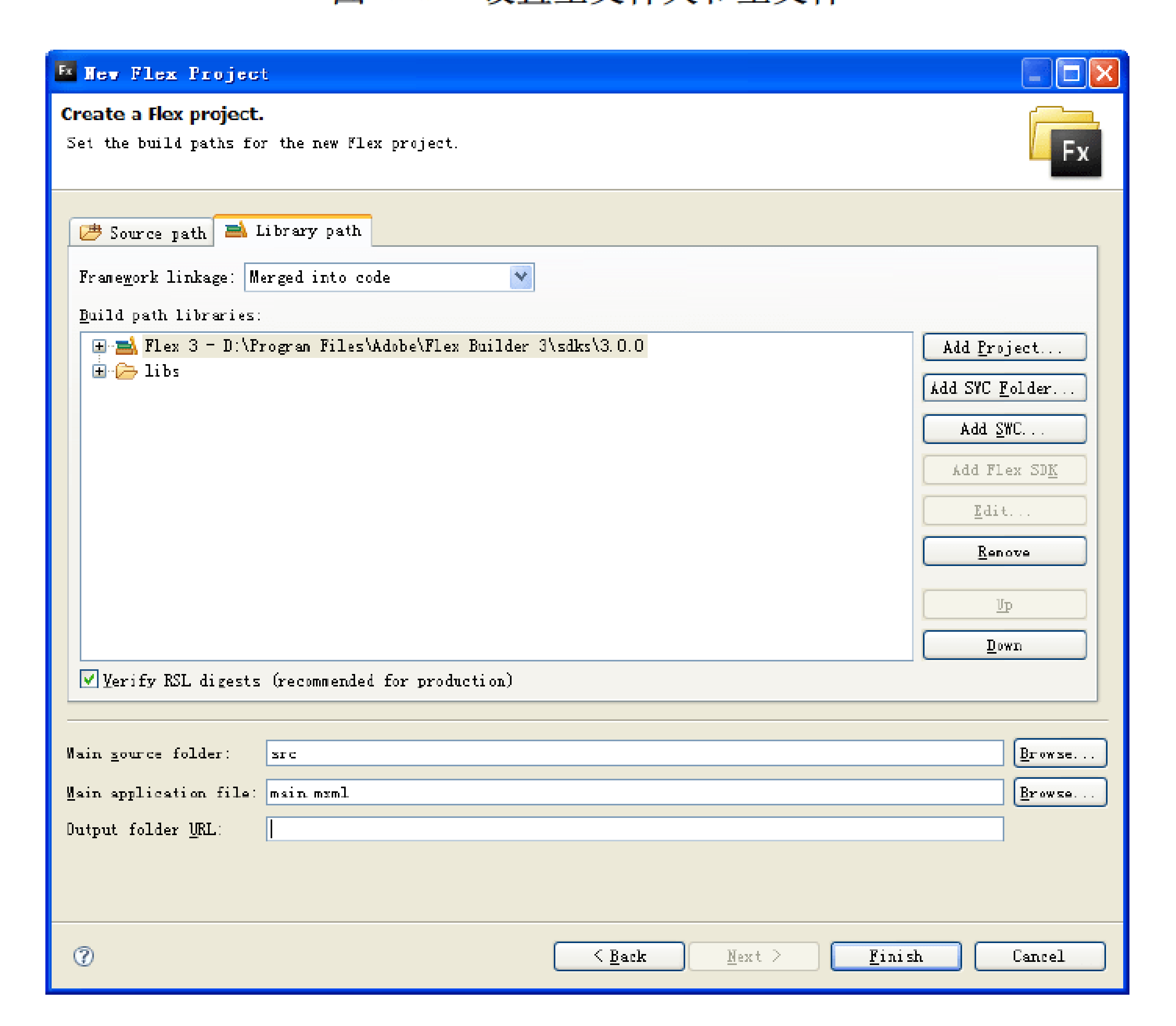


图 2-29　设置 Flex 类库

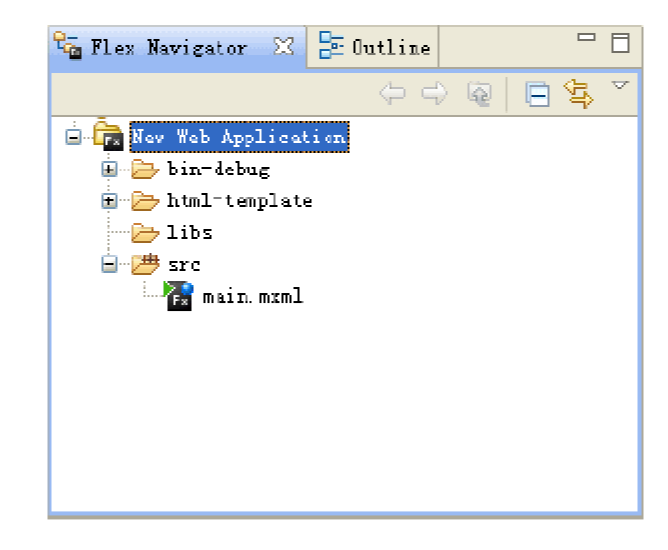


图 2-30　Flex Navigator 窗格

（7）项目创建完成后，打开主文件 main.mxml，切换到设计视图，在 Compotents 窗格中拖曳进来一个 Button 控件和一个 Label 控件，如图 2-31 所示。

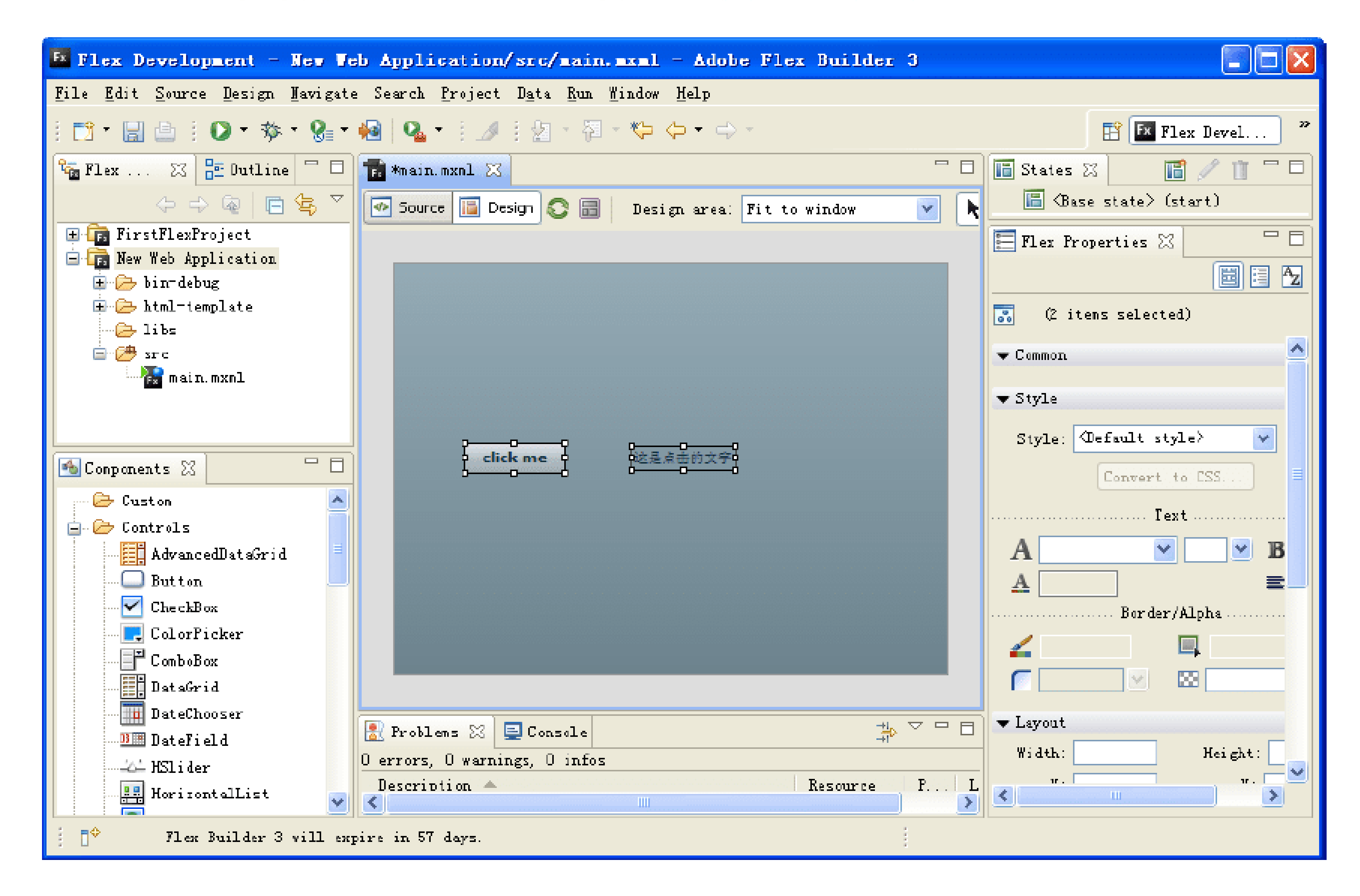


图 2-31　设置页面

（8）设置 Label 控件的属性，设置该 Label 控件默认不显示。具体代码如下所示：

```
<mx:Label x="170" y="61" text="这是点击的文字" visible="false" id="L1"/>
```

（9）设置 Button 按钮的 click 事件为 click="L1.visible='true'"。当用户单击该按钮时，显示 Label 控件。具体的显示效果如图 2-32 所示。

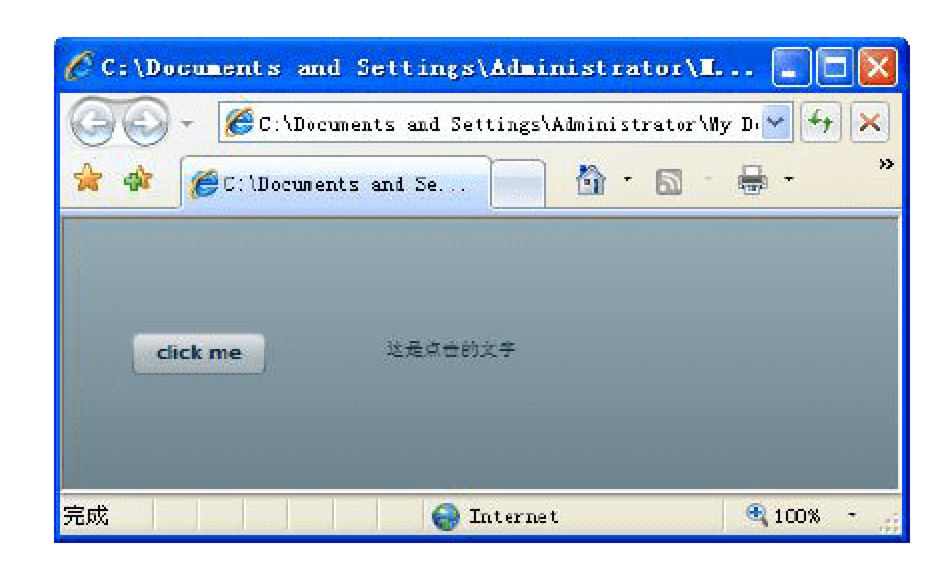


图 2-32　显示 Label 控件

2. Desktop application

Desktop application 项目是桌面应用程序，其创建方式和开发环境与 Web application 项目非常相似。在这里就不再演示创建完整的 Desktop application 项目的具体流程，只展示在创建和开发过程中与 Web application 项目不同的地方。

在创建 Flex 项目时，在 Application type 下面选择 Desktop application，如图 2-33 所示。

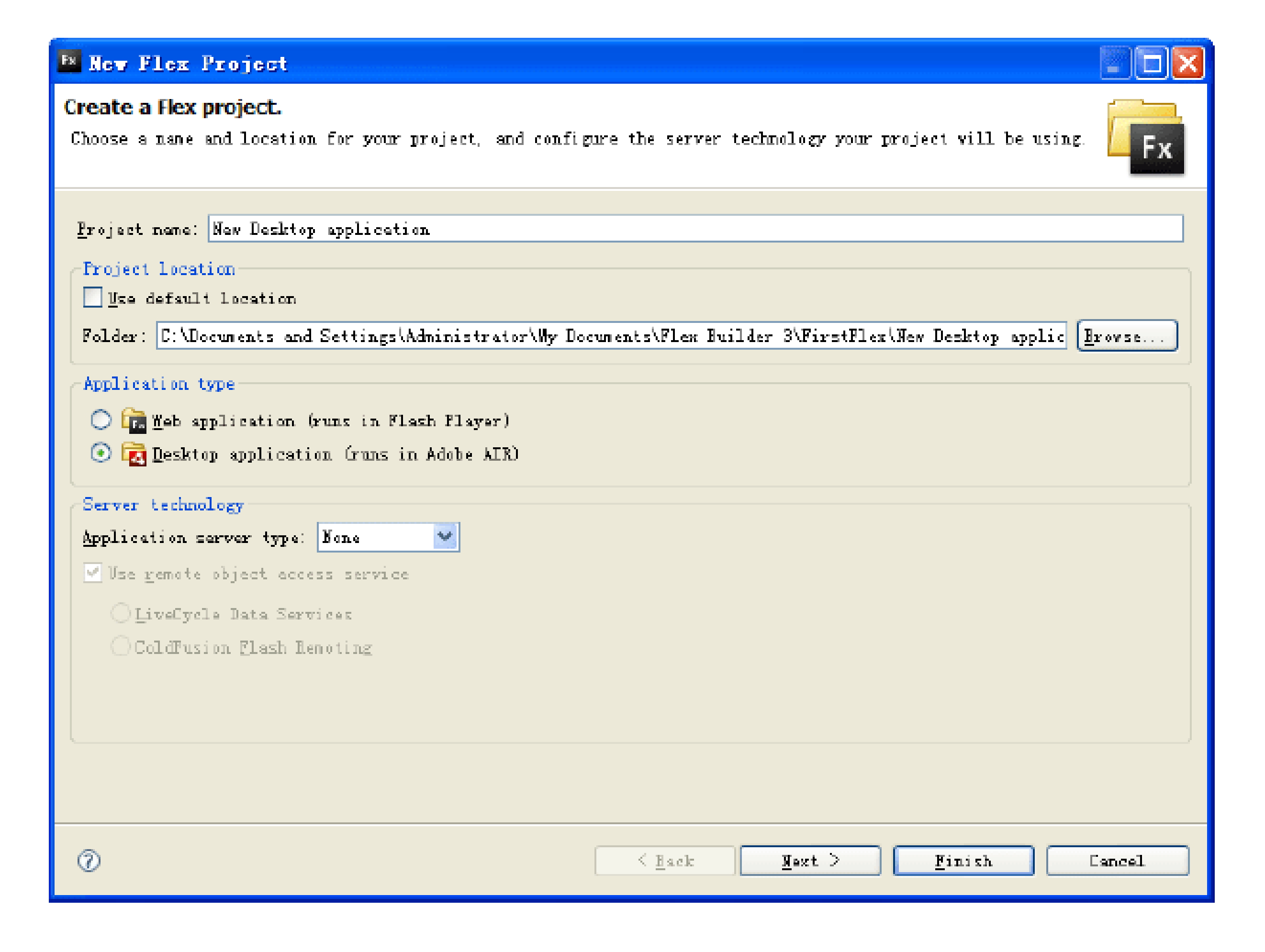


图 2-33　创建 Desktop application 项目

再配置项目的主程序文件夹和文件页面，Desktop application 项目的主窗体还有一个 Application ID 选项，如图 2-34 所示。

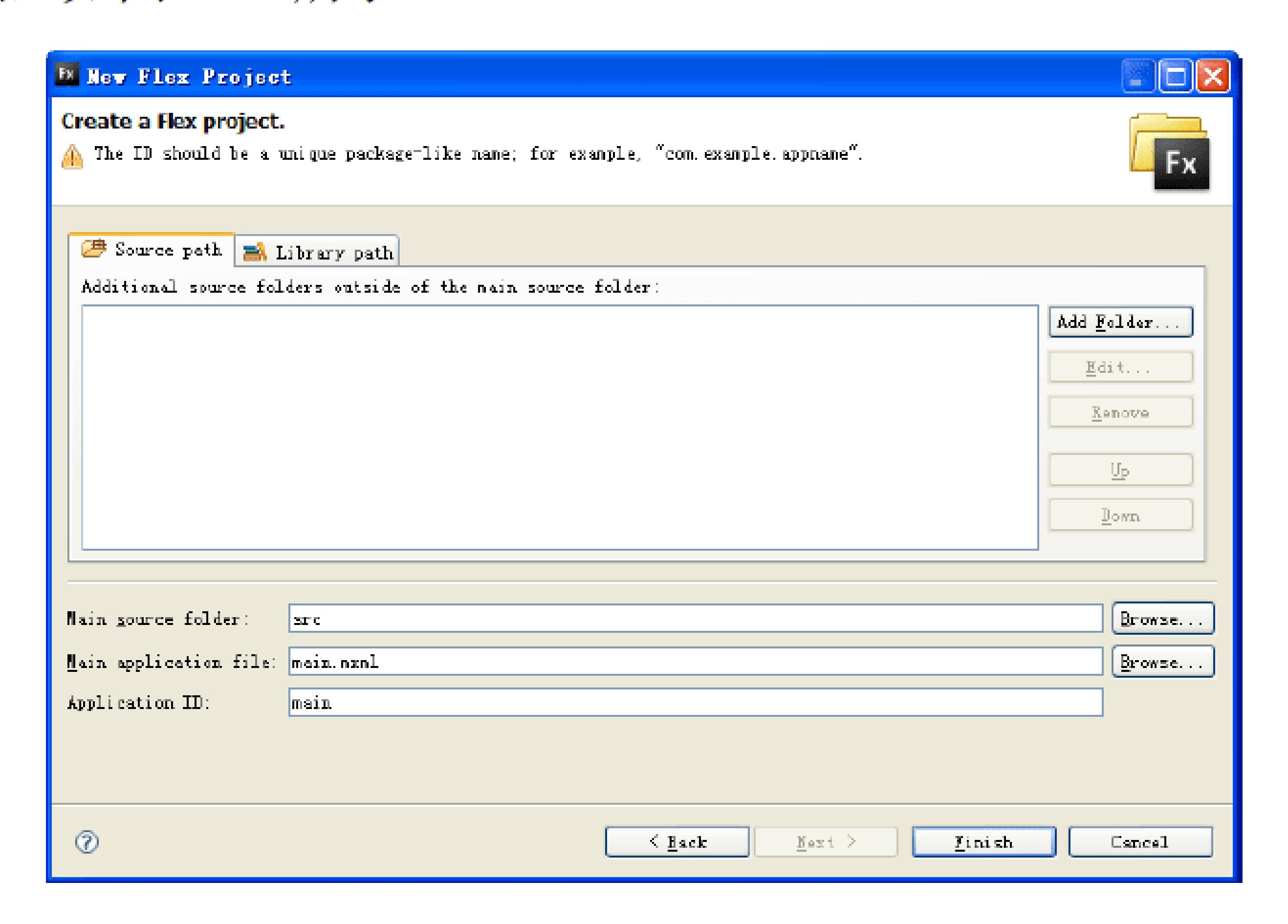


图 2-34　配置主文件的 ID

创建完成后，在 Flex Navigator 窗格中，对比两种类型项目的内容，Desktop application 项目少了 html-template 文件夹，多了一个 main-app.xml 文件，如图 2-35 所示。

最后来看一下 Desktop application 项目的运行效果，如图 2-36 所示。

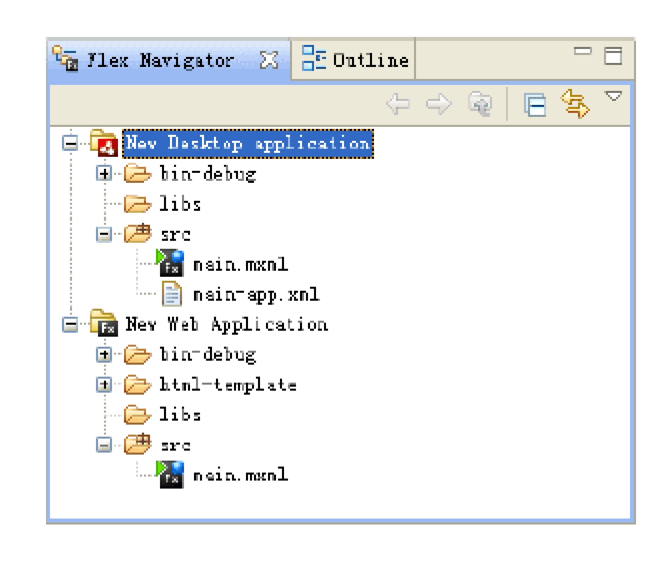


图 2-35　Flex Navigator 窗格

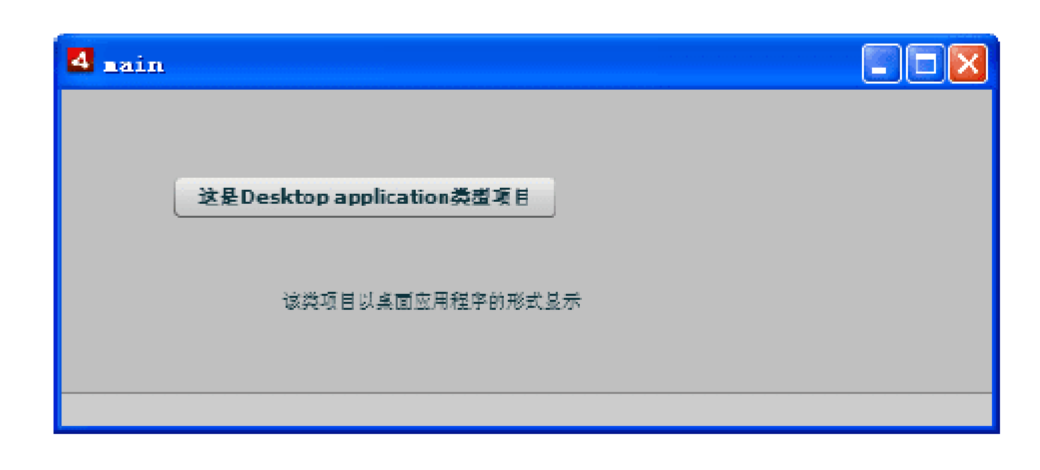


图 2-36　Desktop application 项目的运行效果

2.4.2　ActionScript Project

ActionScript Project 项目与 Flex Project 项目不同，因为它根本不包含 Flex 框架。ActionScript 项目依赖 Flash 基础代码中的核心 ActionScript 类，并且不允许访问 Flex 框架中的任何组件。下面创建一个 ActionScript Project 项目，具体过程如下所示。

（1）选择 File|New | ActionScript Project 命令，如图 2-37 所示。

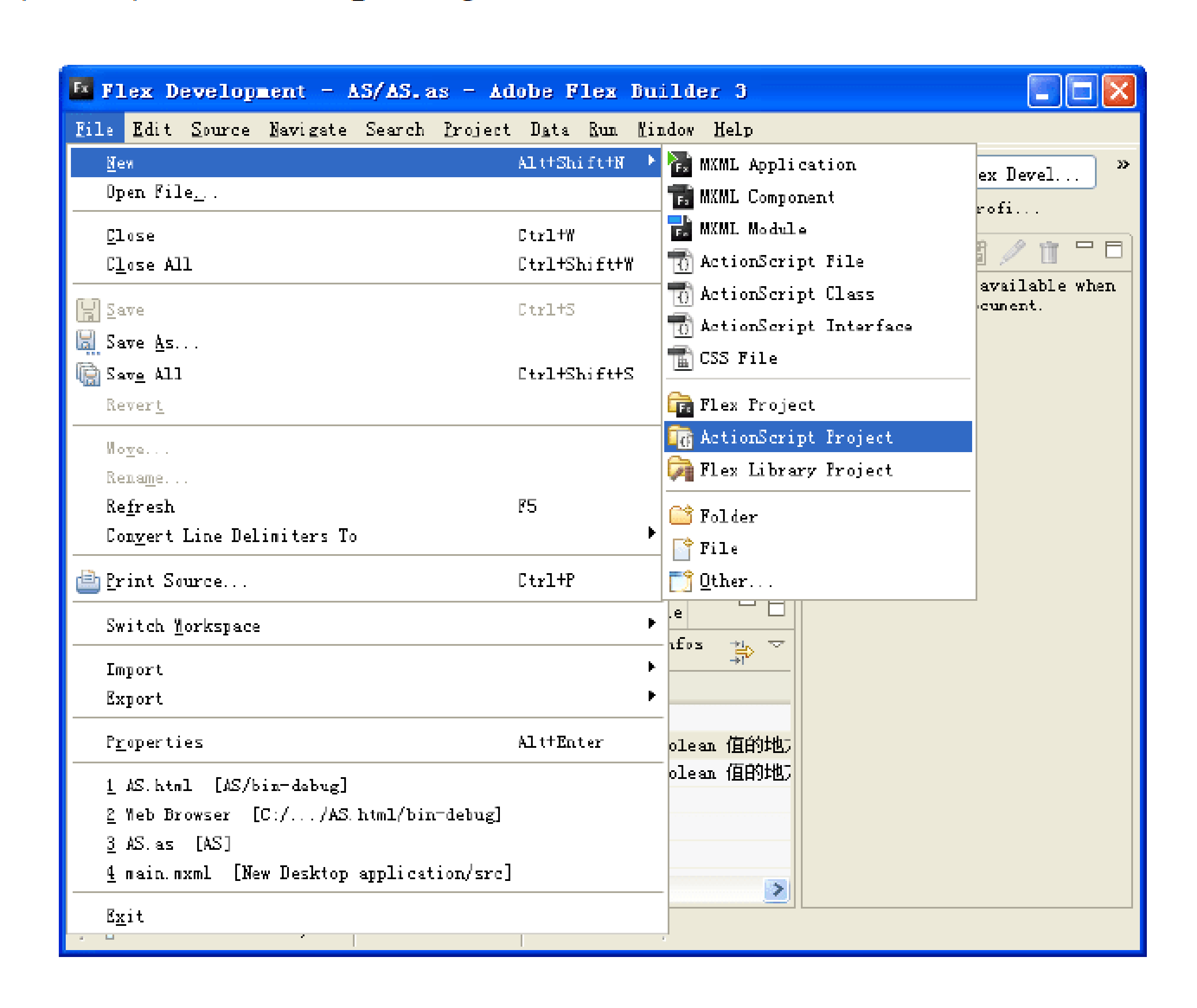


图 2-37　创建 ActionScript Project

（2）输入项目名称 AS，选择该项目的存放路径，如图 2-38 所示。

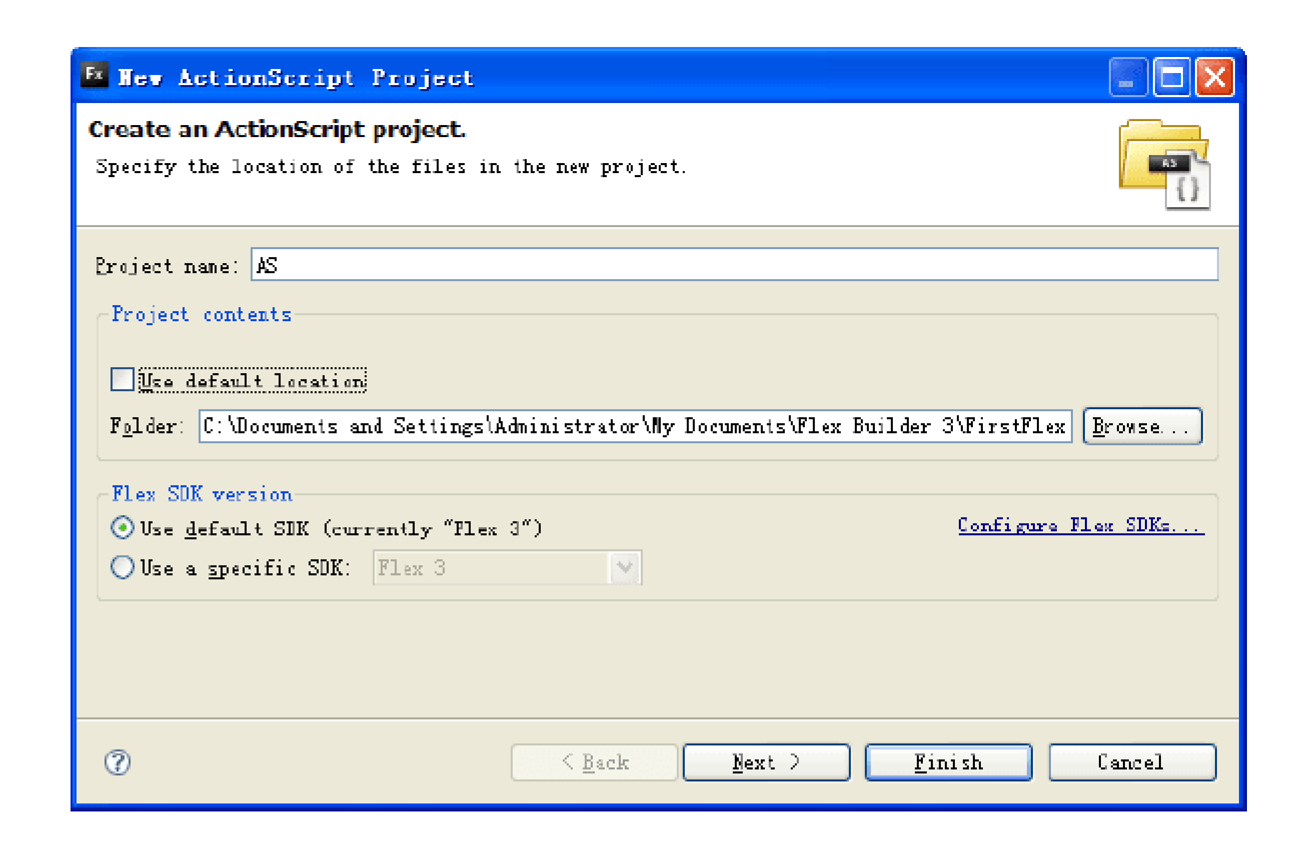


图 2-38　配置 ActionScript Project 选项

（3）单击 Next 按钮，设置存放 AS 文件的文件夹、主文件的文件名和存放输出文件的文件夹，如图 2-39 所示。

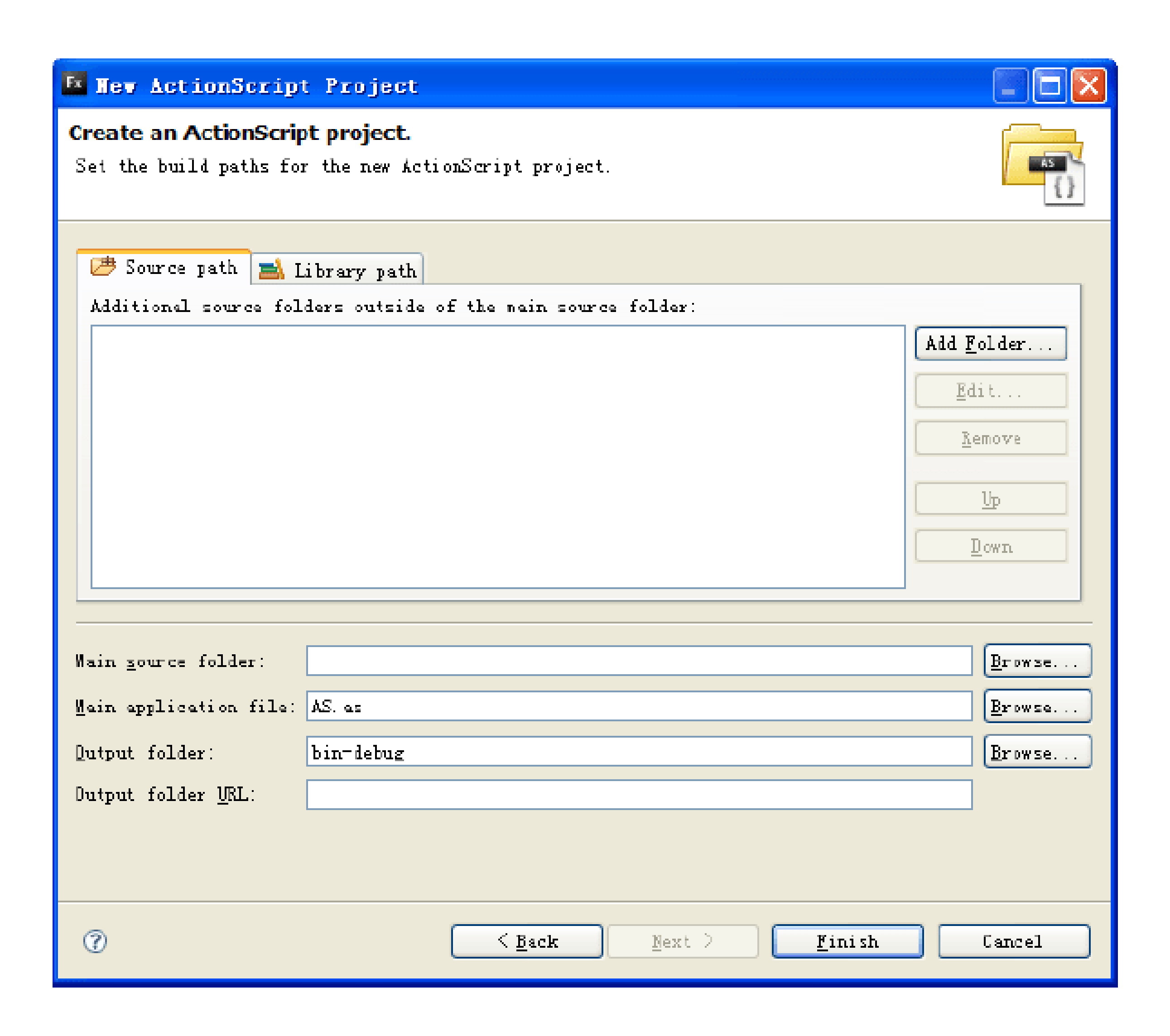


图 2-39　配置主文件和输出文件夹

（4）切换到 Library path 选项卡。配置 ActionScript Project 项目的类库文件存放位置，如图 2-40 所示。

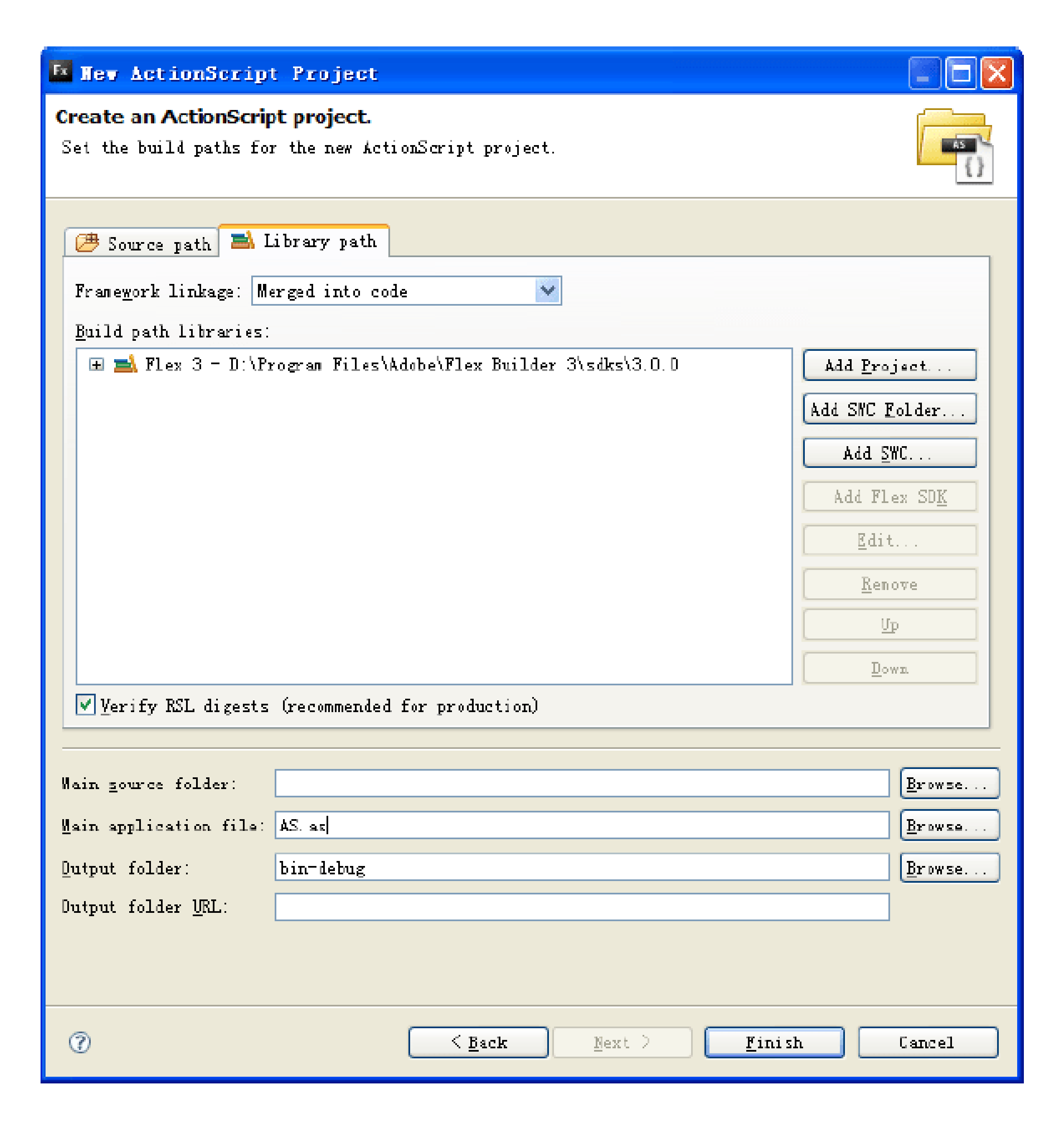


图 2-40　配置类库

（5）单击 Finish 按钮，完成项目的创建。在 Flex Navigator 窗格中，显示了 ActionScript Project 项目的所有内容，如图 2-41 所示。

（6）ActionScript 项目创建后含有一个主文件 AS.as，并且不允许访问 Flex 框架中的任何组件。该文件的初始代码如下所示。

```
package {
    import flash.display.Sprite;
    public class AS extends Sprite
    {
        public function AS()
        {
        }
    }
}
```

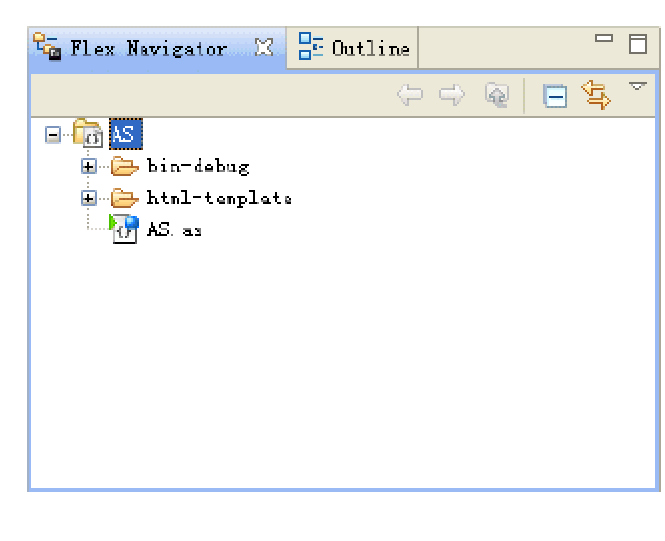


图 2-41　项目内容

ActionScript 项目创建完成后，就可以在主 AS 文件中使用 ActionScript 语言进行具体的功能设计。ActionScript 脚本语言的具体语法基础，将在后面的章节中详细介绍，这里就不进行具体的设计。

2.4.3　Flex Library Project

创建一个 Flex Library Project，不需要主 MXML Application 文件，它不编译到 SWF 文件中，而是改为编译到 swc 文件中，并且可以被用作其他的应用或者是作为运行时共用的库资源。创建 Flex Library Project 可以将自定义的组件保存到一个项目中，并打包成 swc 库文件供其他应用程序调用。

下面创建一个 Flex Library Project，具体步骤如下所示。

（1）选择 File|New | Flex Library Project 命令，如图 2-42 所示。

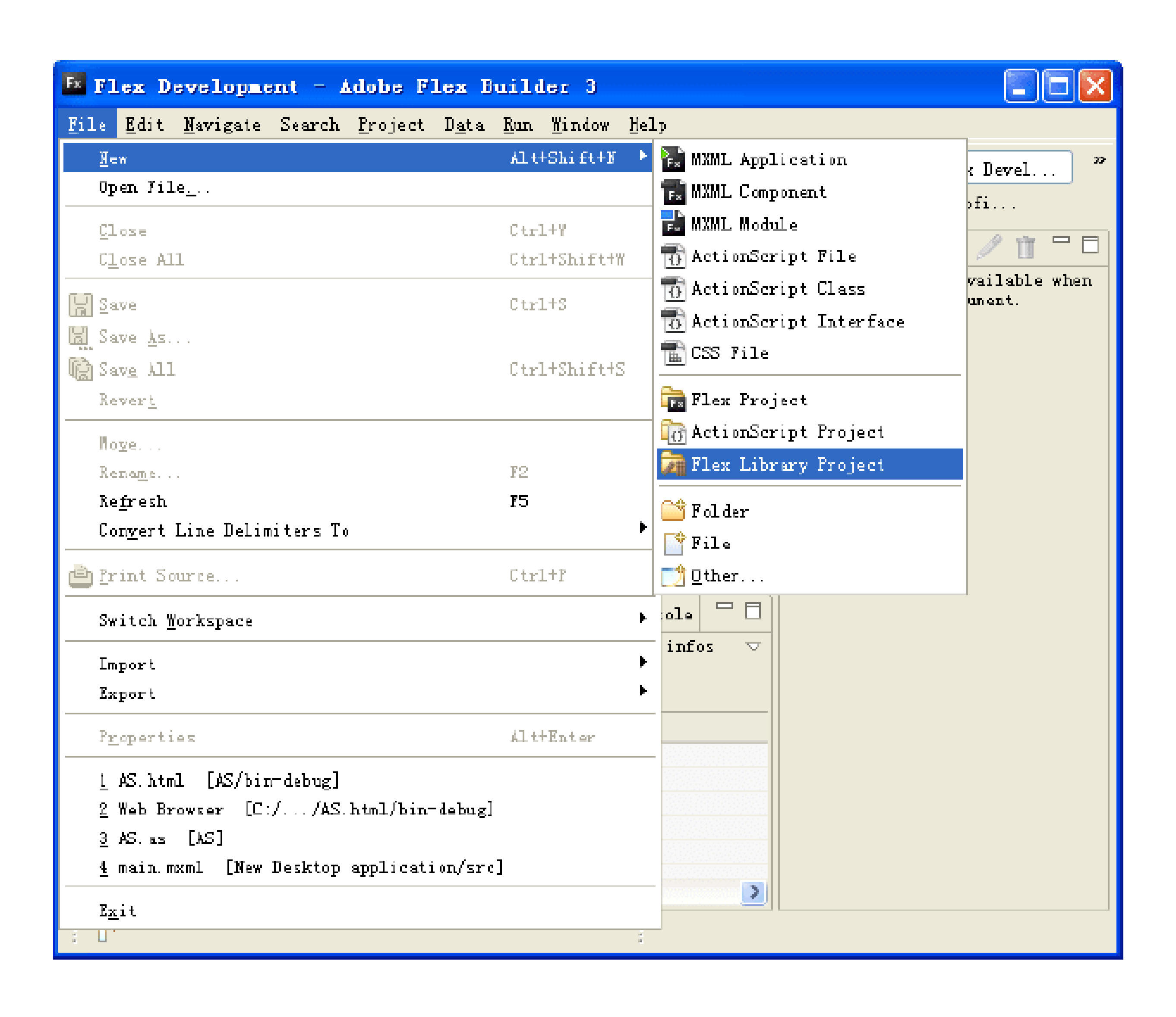


图 2-42　创建 Flex Library 项目

（2）输入项目名称 NewLib，选择该项目的存放路径，并且启用 Include Adobe AIR Libraries 选项，该选项用于包含原有的库文件，如图 2-43 所示。

图 2-43　配置 Flex Library 项目选项

（3）单击 Next 按钮，配置剩余选项。设置存放文件的文件夹为 src，存放输出文件的文件夹为 bin，如图 2-44 所示。

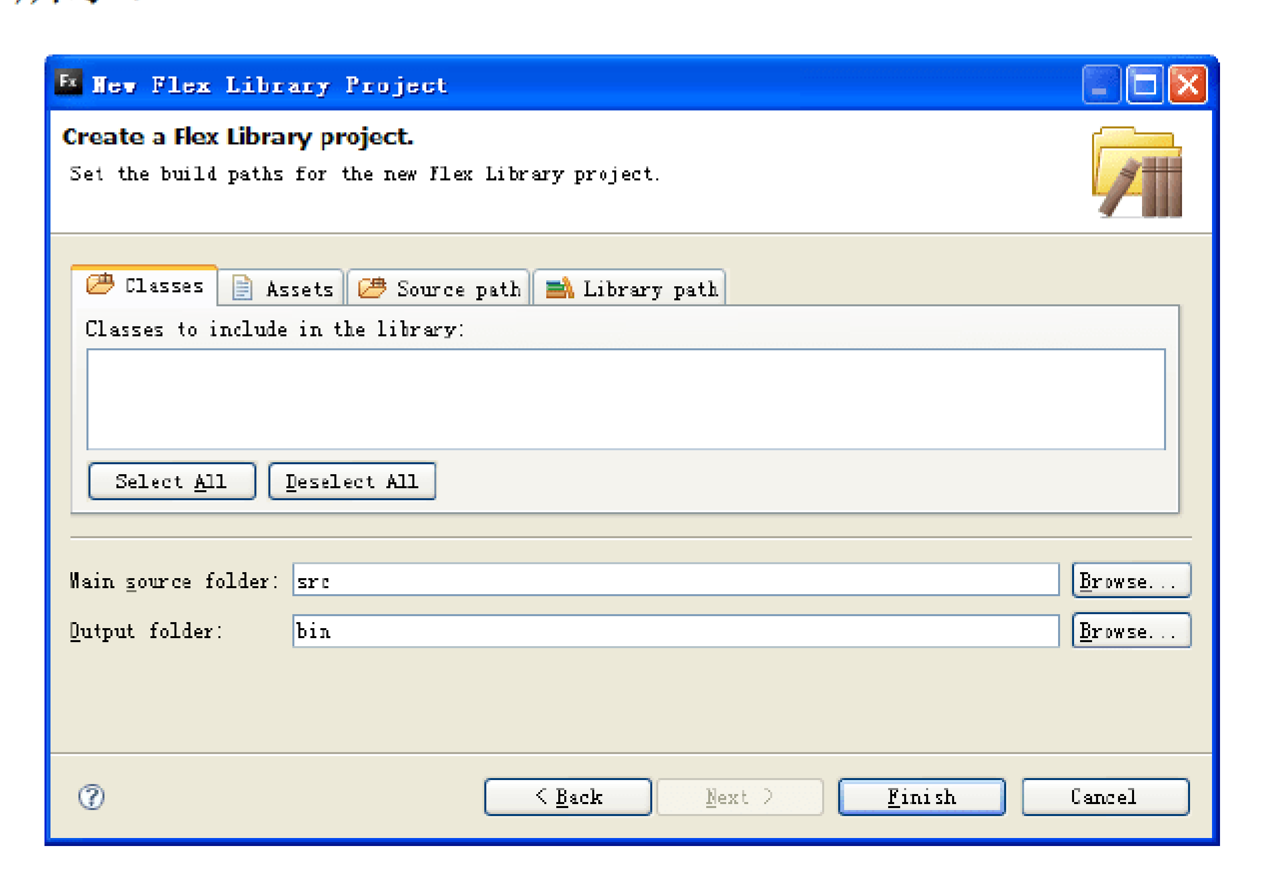


图 2-44　设置主文件夹和输出文件夹

（4）单击 Finish 按钮，完成 Flex 库项目的创建。创建完成后，就可以在 src 文件夹下创建用户自定义的组件。

2.5　Flex Builder 3 中的常用快捷键

前面介绍了 Flex Builder 3 的工作环境和调试、运行 Flex 3.0 程序的一些基本知识。本节将介绍 Flex Builder 3 中常用的快捷键。

一般来说，使用快捷键比使用鼠标来得更快。在 Flex Builder 3 中，选择顶部菜单中的 Help|Key Assist 命令，就可以看到所有的快捷键列表，如图 2-45 所示。

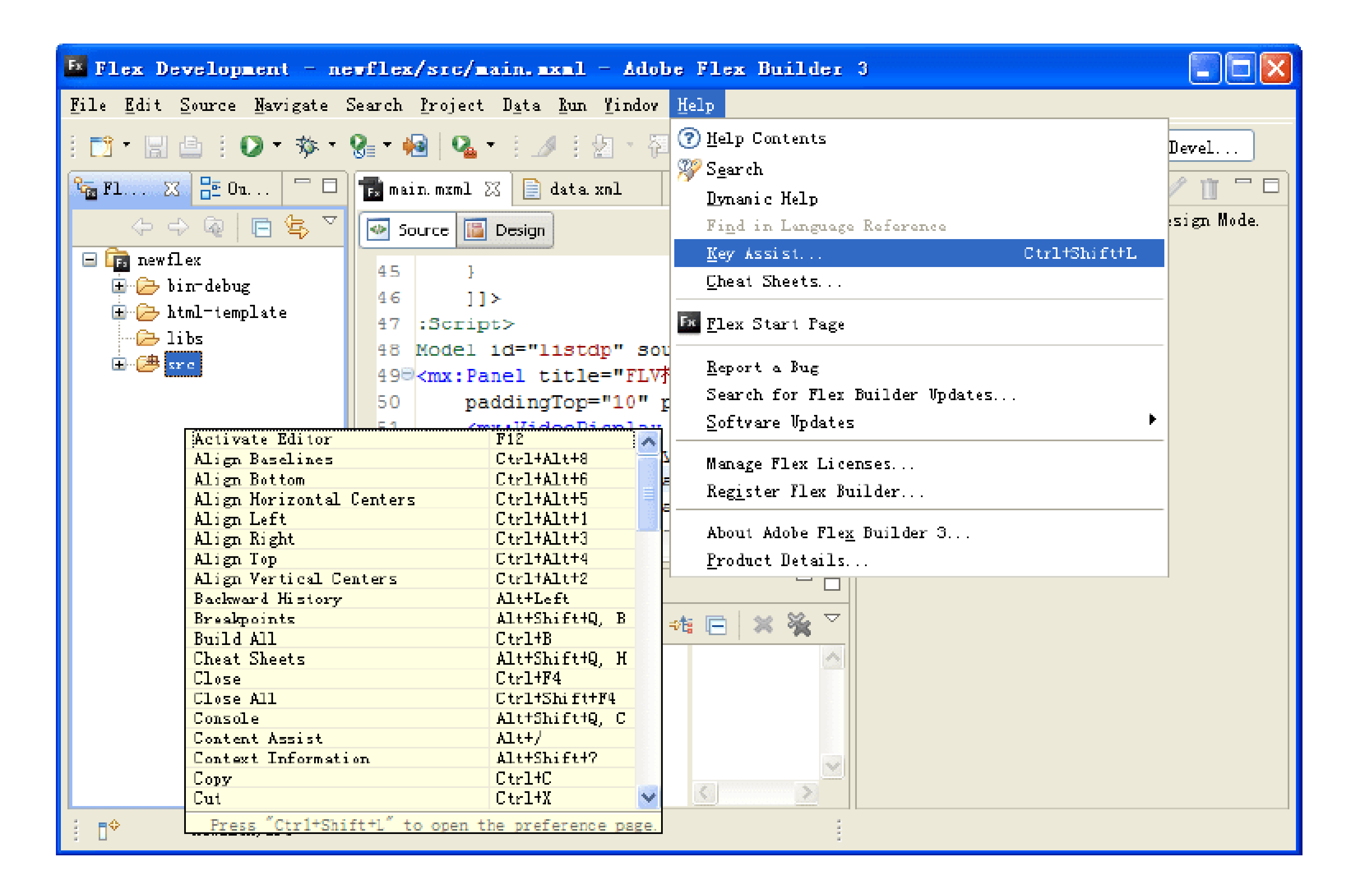


图 2-45　查看快捷键

下面列出了一些常用的快捷键，如表 2-1 所示。

表 2-1　常用快捷键

快捷键	说明
Ctrl+N	新建一个文档
Ctrl+S	保存正在编辑的文档
F12	激活编辑区
F11	执行程序调试动作
Ctrl+F11	编译并运行程序
Ctrl+C	将选择的文本或者文件复制到剪贴板上
Ctrl+D	删除当前所在行
Ctrl+V	将剪贴板中的文本或者文件复制到目标处
Alt+/	调出跟进的代码提示
Alt+UP	将所在行上移

2.6　使用 Flex 帮助文档

在 Flex Builder 3 的开发环境中，Help 菜单除了能够完成前面介绍过的查看所有的快捷键信息外，还有其他重要的功能。例如，调出帮助文档、查找特定内容、更新软件等。本节将介绍如何使用 Flex 帮助文档的方法。

在开发过程中，用户不可避免会遇到一些这样那样的语法问题或者错误信息。作为一个合格的开发人员，学会使用 Flex 帮助文档去解决问题十分必要。Flex 帮助文档提供了最全面的 Flex 内容，其主页面如图 2-46 所示。

Flex 帮助文档在安装 Flex Builder 3 时默认安装，开发者可以在文本框中输入想要查询的任何内容，Flex 将返回所有符合查询条件的内容。在文档的左边列表中，列出了这些符合查

询条件的内容，右边框架内用于显示每一条内容的详细信息，如图 2-47 所示。

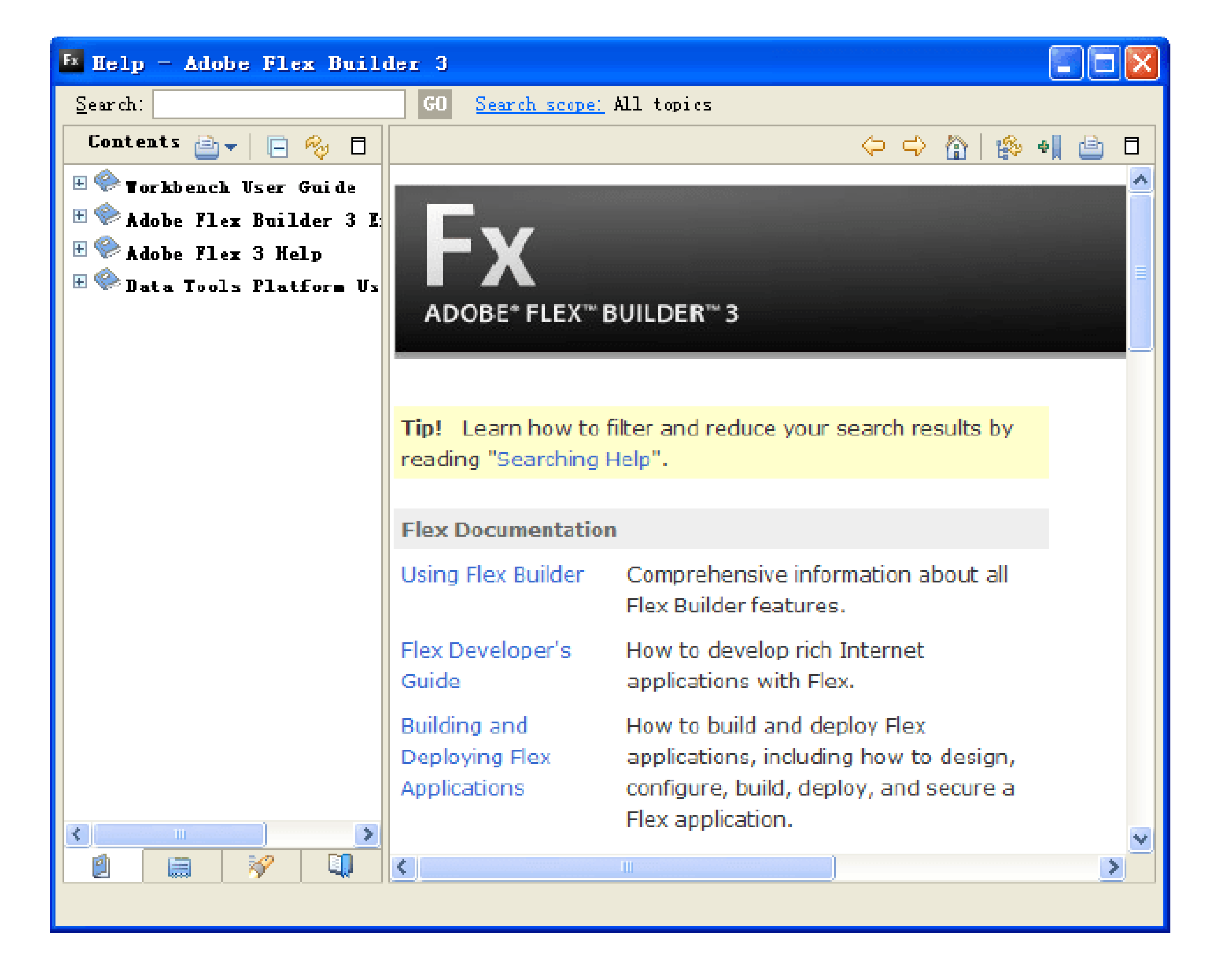


图 2-46　Flex 帮助文档

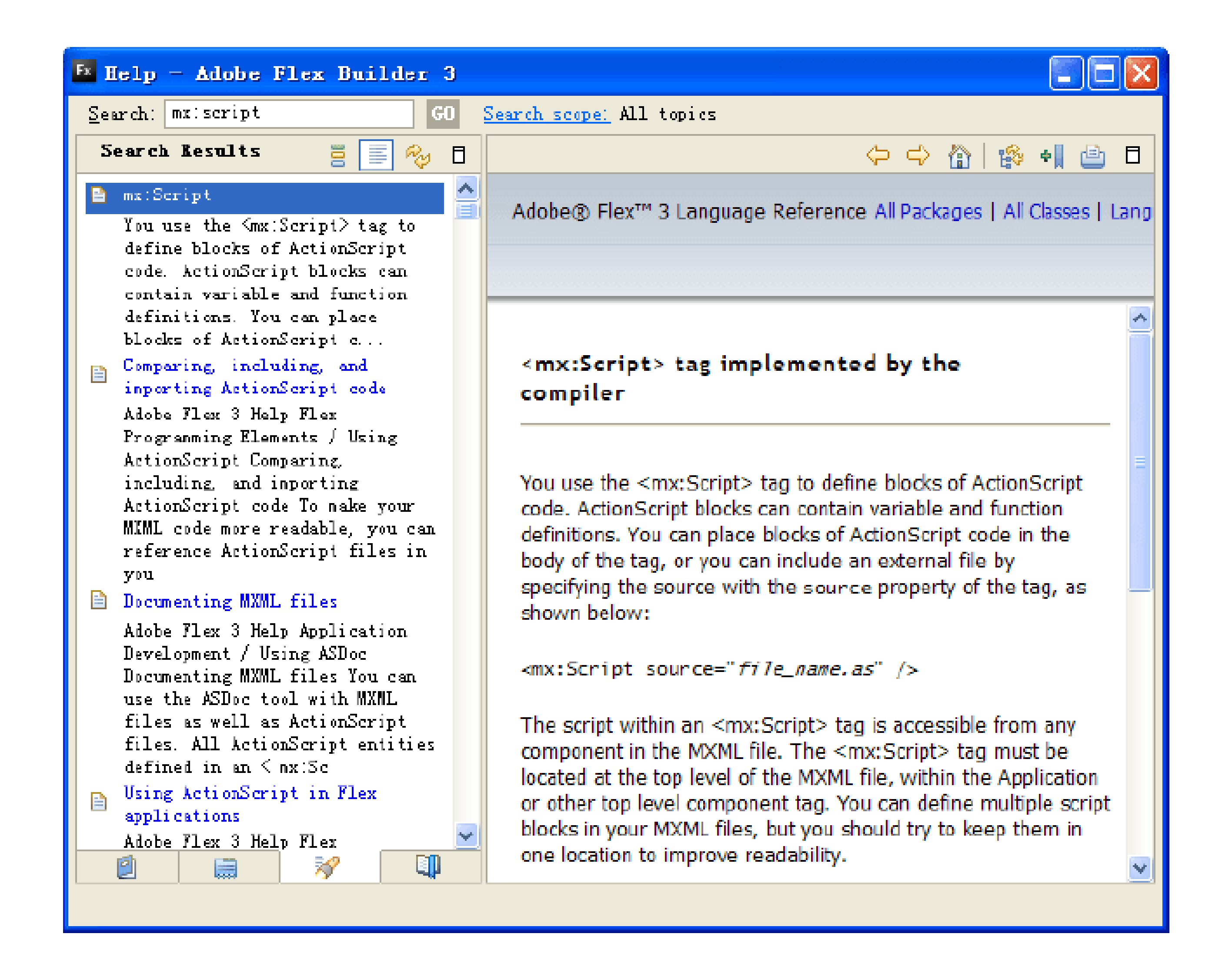


图 2-47　使用 Flex 帮助文档查询

第3章 ActionScript 3.0 语法



ActionScript 3.0 代码比旧版本的运行速度要快上 10 倍，并且好学、容易调试，因为这种语言依附于标准的编程语言（基于 JavaScript）ECMAScript，使用新的、轻量的、可轻松设置外观的界面组件，为 ActionScript 3.0 创建应用程序。使用绘图工具，便可以修改组件的外观，而不需要进行编码。本章将对 ActionScript 3.0 的基础语法进行介绍，包括 ActionScript 3.0 中的常量、变量、数据类型、运算符以及控制语句的使用等。

学习目标 | Objective

- 掌握 ActionScript 3.0 中的常量和变量
- 掌握 ActionScript 3.0 中的数据类型
- 熟悉 ActionScript 3.0 中对数据类型的检查
- 掌握数据类型的转换方法
- 掌握 ActionScript 3.0 中运算符的使用方法
- 掌握各种流程控制语句

3.1 常量和变量

所有的程序都要对数据进行处理。在对数据进行处理的过程中，需要使用一个容器将其中的数据保存起来，这样就可实现对容器中的数据进行再处理。根据容器中保存的数据，在程序运行的过程中是否可以修改，容器可分为两种：常量和变量。

3.1.1 常量

常量（Constant），是一种恒定的或者不可变的数值或者数据项。某些时候，虽然声明了一个变量，但却不希望这个数值被修改，这种永不会被修改的变量，统称为常量。在 ActionScript 3.0 中常量可以分为系统自带常量和用户自定义常量两种。

1. 系统自带常量

系统自带常量又包括顶级常数或全局常数及类中的公共常量，其中全局常数在每个脚本

中都可用，并且对文档中的所有时间轴和作用域都可见，包括以下 4 个常量。

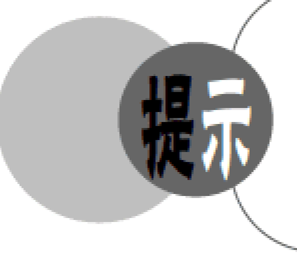
Infinity 该常量是一个 Number 数据类型的数据，表示正 Infinity 的特殊值。此常数的值与 Number.POSITIVE_INFINITY 相同。例如正数除以 0 的结果为 Infinity。

-Infinity 该常量是一个 Number 数据类型的数据，表示负 Infinity 的特殊值。此常数的值与 Number.NEGATIVE_INFINITY 相同。例如负数除以 0 的结果为-Infinity。

NaN Number 数据类型的一个特殊成员，用来表示“非数字”（NaN）值。当数学表达式生成的值无法表示为数字时，结果为 NaN。下表描述了生成 NaN 的常用表达式。

表 3-1 生成 NaN 的常用表达式

0 除以 0
负数的平方根
在有效范围 0 到 1 之外的数字的反正弦值
Infinity 减去 Infinity
Infinity 或-Infinity 除以 Infinity 或-Infinity
Infinity 或-Infinity 乘以 0



NaN 值不是 int 或 uint 数据类型的成员。NaN 值不被视为等于任何其他值（包括 NaN），因而无法使用等于运算符测试一个表达式是否为 NaN。若要确定一个数字是否为 NaN，使用 isNaN()函数。

undefined 一个适用于尚未初始化的无类型变量或未初始化的动态对象属性的特殊值。在 ActionScript 3.0 中，只有无类型变量才能采用 undefined，这在 ActionScript 1.0 和 ActionScript 2.0 中并非如此。例如，以下两个变量都是 undefined，因为它们都未类型化或初始化：

```
var foo;
var bar:*;
```

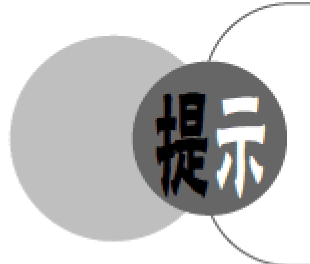
undefined 值还适用于动态对象的未初始化或未定义的属性。例如，如果某对象是 Object 类的一个实例，则除非向该属性赋予值，否则动态添加的任何属性的值将为 undefined。

将 undefined 用于不同的函数时，结果也不相同，表 3-2 显示了不同类型的函数返回的 undefined 值。

表 3-2 不同类型的函数返回的 undefined 值

函数类型	undefined 值
String（undefined）	"undefined"（undefined 转换为字符串）
Number（undefined）	NaN
Int（undefined）和 uint（undefined）	0
Object（undefined）	新的 Object 实例

向类型变量赋予值 undefined 时，该值将转换为该数据类型的默认值。



不要混淆 undefined 和 null。使用等于运算符（==）对 null 和 undefined 进行比较时，它们的比较结果为相等。但是，使用全等运算符（===）对 null 和 undefined 进行比较时，它们的比较结果为不相等。

类中的公共常量在使用时，需要指定所在的类，例如 Math.PI 就是 Math 类中的一个常量，不能直接使用 PI。

2．用户自定义常量

用户自定义常量，是在程序开发时，用户通过 const 关键字定义的变量，该变量只能赋一次值。语法格式如下所示：

```
const identifier=value
```

可以通过在数据类型前追加一个冒号（:）字符来严格限定用户自定义常量的类型。

以下示例显示，如果尝试多次对一个用户自定义常量赋值，就会发生错误。

```
const MIN AGE:int=21;
MIN_AGE=18;//发生错误
```

3.1.2　变量

变量是在代码中描述或容纳动态改变的值或数据的名称。当在 ActionScript 3.0 中声明变量时，需要使用 var 关键词，如下所示。

```
var myVariableName; //需要使用 var
```

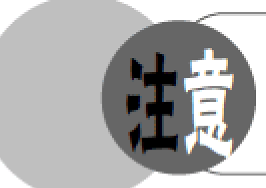
从 Flash 5 版本开始，var 关键词就已经可用了，但是在现在的 ActionScript 3.0 中它是必须的。例外的情况是在定义动态对象实例的变量时，如下所示。

```
myDynamicObject.newVar=value;//不需要 var
```

在上面的例子中，newVar 是在 myDynamicObject 对象中定义的一个变量，没有使用 var 关键词。实际上，var 关键词不在复杂引用中应用，或是任意需要点语法的引用中，或是使用[]引用的变量中。例如函数的参数等。

当定义一个变量时必须使用数字、字符、$符号或下划线_来命名变量，不能使用数字开头的名字作为变量名称。如“bits32”、“_bits32”和“$bits32”都是合法的，“32bits”是不合法的，因为名称的第一个字符使用了数字。

需要确认所创建的变量名，和现有的代码中的变量和代码内置的变量没有冲突。例如，当在时间轴上书写代码时，正在定义一个影片剪辑的实例名称，如果试图将其定义成与 Flash 内置的变量 MovieClip 名称相同，将会得到一个错误。同样，不能使用全局对象和函数名称，如 Array、XML 或是 trace 等。



从 Flash Player 7 之后，ActionScript 开始区分大小写，因此变量 Name 和 name 是不同的。

ActionScript 3.0 和早期版本比较起来，还有一个新的地方，就是某一个变量的定义，只能在代码范围内或是时间轴代码上使用一次 var，从另一个角度说，如在一段代码的顶端声明了变量 x，那么不能在下面代码中的 x 变量前使用 var 关键词。如下所示的代码就会产生错误。

```
var x = value; //正确
…
var x = differentValue; //错误：只能使用一次 var 关键词
```

当在 Flash 的时间轴上定义变量时，它会应用在整个时间轴上，而不只是当前的帧。

当使用 var 关键词定义变量时，可以为其指定数据类型。一个变量的数据类型，描述该变量存放的是哪种类型的数据。这个特性是从 ActionScript 2.0 开始的，一直延续到现在的 ActionScript 3.0。例如，如果定义变量 x 是一个数值，那么可以为它指定为数值型，如下所示：

```
var x:Number; //变量 x 将用于承载数值型数据
```



为变量指定类型是个好的习惯，因为它可以引导更好、更快地进行错误检查。

3.2 数据类型

“数据类型”用来定义一组相似的值。在 ActionScript 3.0 中，类型信息在运行时保留，并可用于多种目的。例如，Flash Player 9 在运行时要执行类型检查，增强了系统的类型安全性。类型信息还用于指定变量的储存、检索等形式，从而提高了系统的性能并减少了内存使用量。

ActionScript 3.0 的数据类型分为基本数据类型和复合数据类型。例如，Boolean 数据类型所定义的值仅包含两个可能的值：true 和 false。除了 Boolean 数据类型外，ActionScript 3.0 还定义了其他几个基本数据类型，如 String、Number 和 Null 等。也可以使用类或接口来自定义一组值，从而定义自己的数据类型。本章节将详细讲述数据类型的相关知识。

3.2.1 基本数据类型

基本数据类型包括 Boolean、int、uint、Number、String、Null 和 void。这些类型都属于值类型，不需要使用 new 关键字声明的变量，其中 int 和 uint 是 ActionScript 3.0 中新增的数据类型，下面分别介绍这 7 种数据类型。

1. Boolean

Boolean 数据类型，实际上就是 Boolean 对象，可以使用 true 或 false（用于进行逻辑运算）两个值中一个值，其他任何值都是无效的。已经声明但尚未初始化的 Boolean 型变量的默认值

是 false。

若要创建 Boolean 对象，可以使用构造函数、全局函数，或赋予值。在 ActionScript 3.0 中，上述 3 种技术功效相同。（这与 JavaScript 不同，JavaScript 中的 Boolean 对象与 Boolean 原始类型不同。）如下代码中所示，3 种定义变量的方法效果是相同的。

```
var flag:Boolean = true;
var flag:Boolean = new Boolean(true);
var flag:Boolean = Boolean(true);
```

2. int

int 数据类型在内部存储为 32 位整数，它包含一组介于–2 147 483 648 和 2 147 483 647 之间的整数（包括–2 147 483 648 和 2 147 483 647），int 数据类型的变量，默认值是 0。int 数据类型变量的定义语法如下所示。

```
var myint:int = 1234;
```

早期的 ActionScript 版本仅提供 Number 数据类型，该数据类型既可用于整数又可用于浮点数。在 ActionScript 3.0 中，现在可以使用 32 位带符号整数和无符号整数。

如果要定义的变量不会出现浮点数，那么，使用 int 数据类型来代替 Number 数据类型会更快、更高效。反之，应使用 Number 数据类型。

3. uint

uint 数据类型在内部存储为 32 位无符号整数，它包含一组介于 0 和 4 294 967 295 之间的整数（包括 0 和 4 294 967 295）。对于大于 uint 的最大值的整数值，应使用 Number 数据类型。uint 数据类型的变量的默认值是 0。uint 数据类型变量的定义语法如下所示。

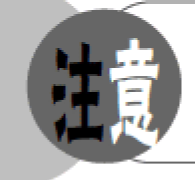
```
var myuint:uint = 1234;
```

uint 数据类型可用于要求非负整数的特殊情形，例如，必须使用 uint 数据类型来表示像素颜色值，因为 int 数据类型有一个内部符号位，该符号位并不适合于处理颜色值。

4. Number

在 ActionScript 3.0 中，Number 数据类型可以表示整数、无符号整数和浮点数。但是，为了尽可能提高性能，应将 Number 数据类型仅用于表示浮点数，或者用于表示 int 和 uint 类型可以存储的、大于 32 位的整数值。Number 数据类型变量的定义语法如下所示。

```
var num:Number = 315003;
```



要存储浮点数，数字中应包括一个小数点。如果省略了小数点，数字将存储为整数。

Number 数据类型使用由 IEEE 二进制浮点算术标准（IEEE-754）指定的 64 位双精度格式。此标准规定如何使用 64 个可用位来存储浮点数。其中的 1 位用来指定数字是正数还是负数，

11 位用于存储指数，它以二进制的形式存储，其余的 52 位用于存储“有效位数”（又称为“尾数”），有效位数是 2 的 N 次幂，N 即前面所提到的指数。

可以将 Number 数据类型的所有位都用于存储有效位数，也可以将 Number 数据类型的某些位用于存储指数，则后者可存储的浮点数比前者大得多。

Number 类型可以表示的最大值和最小值，分别存储在 Number 类的名为 MAX_VALUE 和 MIN_VALUE 的静态属性中。

```
Number.MAX_VALUE==1.79769313486231e+308
Number.MIN_VALUE==4.940656458412467e-324
```

尽管这种数据类型范围很大，但代价是此范围的精度有所降低。Number 数据类型使用 52 位来存储有效位数，因此，那些要求用 52 位以上的位数才能精确表示的数字（如分数 1/3）将只是近似值。如果应用程序要求小数达到绝对精度，则需要使用实现小数浮点算术（而非二进制浮点算术）的软件。

如果用 Number 数据类型来存储整数值，则仅使用 52 位有效位数。Number 数据类型使用 52 位和一个特殊的隐藏位，来表示介于–9 007 199 254 740 992 和 9 007 199 254 740 992 之间的整数。

Flash Player 不但将 NaN 值用作 Number 类型的变量的默认值，而且还将其用作应返回数字、却没有返回数字的任何运算的结果。例如，如果尝试计算负数的平方根，结果将是 NaN。其他特殊的 Number 值包括 Infinity 和–Infinity。

5. String

String 数据类型表示一个 16 位字符的序列。字符串在内部存储为 Unicode 字符，并使用 utf-16 格式。String 数据类型变量的定义语法如下所示。

```
var str:String = "foo";
```

在 ActionScript 3.0 中，字符串是不可改变的值，就像在 Java 编程语言中一样。对字符串值执行运算会返回字符串的一个新实例。用 String 数据类型声明的变量的默认值是 null。虽然 null 值与空字符串（""）均表示没有任何字符，但二者并不相同。

6. Null

Null 数据类型仅包含一个值 null。这是 String 数据类型和用来定义复合数据类型的所有类（包括 Object 类）的默认值。其他基本数据类型（如 Boolean、Number、int 和 uint）均不包含 null 值。如果尝试向 Boolean、Number、int 或 uint 类型的变量赋予 null，则 Flash Player 会将 null 值转换为相应的默认值。



不能将 Null 用作定义变量的数据类型，只能将此类型的 null 值赋予某一变量。

7. void

void 数据类型仅包含一个值 undefined。在早期的 ActionScript 版本中，undefined 是 Object

类实例的默认值。在 ActionScript 3.0 中，Object 实例的默认值是 null。如果尝试将值 undefined 赋予 Object 类的实例，Flash Player 会将该值转换为 null。只能为无类型变量赋予 undefined 这一值。无类型变量是指缺乏类型注释或者使用星号（*）作为类型注释的变量。只能将 void 用作返回类型注释。

3.2.2　复合数据类型

ActionScript 核心类还定义下列复合数据类型：Object、Array、Date、Error、Function、RegExp、XML 和 XMLList。这些数据类型定义的变量，有一个可变的值，包含了对变量实际值的引用，也就是该变量的值所对应的内存地址，即储存的不是值本身，而是值的引用地址。

1. Object

Object 数据类型是由 Object 类定义的。Object 类用作 ActionScript 中的所有类定义的父类。Object 数据类型的定义语法如下所示。

```
var myobject:Object=new Object();
```

ActionScript 3.0 中的 Object 数据类型，与早期版本中的 Object 数据类型存在以下 3 方面的区别。

Object 数据类型不再是指定给没有类型注释的变量的默认数据类型。

Object 数据类型不再包括 undefined 这一值，该值之前是 Object 实例的默认值。

在 ActionScript 3.0 中，Object 类实例的默认值是 null。

在早期的 ActionScript 版本中，会自动为没有类型注释的变量赋予 Object 数据类型。ActionScript 3.0 现在包括真正无类型变量这一概念，因此不再为没有类型注释的变量赋予 Object 数据类型。没有类型注释的变量现在被视为无类型变量。如果希望在代码中明确指定变量为无类型，可以使用星号（*）表示类型，这与省略类型是等效的。下面的代码显示两条等效的语句，两者都声明一个无类型变量 x。

```
var x
var x:*
```

只有无类型变量才能保存值 undefined。如果尝试将值 undefined 赋给具有数据类型的变量，Flash Player 会将值 undefined 转换为该数据类型的默认值。对于 Object 数据类型的实例，默认值是 null，这意味着，如果尝试将 undefined 赋给 Object 实例，Flash Player 会将值 undefined 转换为 null。

2. Array

Array 数据类型是由 Array 类定义的，Array 类可以访问和操作数组。Array 索引从零开始，这意味着数组中的第一个元素为[0]，第二个元素为[1]，依次类推。要创建 Array 对象，可以使用 new Array()构造函数。Array()还可以作为函数被调用。此外，还可以使用数组访问（[]）运算符初始化数组或访问数组元素。

可以在数组元素中存储各种各样的数据类型，包括数字、字符串、对象，甚至是其他数组。可以创建一个多维数组，方法是创建一个索引数组，然后给它的每个元素分配不同的索引数组。

数组是稀疏结构，这意味着可能存在这样的情况：在索引 0 处有一个元素，在索引 5 处有另一个元素，而这两个元素之间的索引位置却是空的。在这种情况下，位置 1 至位置 4 的元素是未定义的，表示这些位置不存在元素。

数组赋值是通过引用而不是通过值进行的。如果将一个数组变量赋值给另一个数组变量，则这两个变量引用同一个数组，如下所示的代码演示了该效果。

```
var oneArray:Array = new Array("a", "b", "c");
var twoArray:Array = oneArray;
twoArray[0] = "z";
trace(oneArray); //输出结果为: z,b,c
```

不要使用 Array 类创建关联数组（也称为哈希），关联数组是包含命名元素而不包含编号元素的数据结构。要创建关联数组可使用 Object 类。虽然 ActionScript 允许使用 Array 类创建关联数组，但不能对关联数组使用 Array 类的任何方法或属性。

3. Date

Date 数据类型由 Date 类定义，该类表示日期和时间信息。若要使用 Date 类，需要使用 new 运算符构造一个 Date 实例。Date 数据类型变量的定义语法如下所示。

```
var myDate1:Date = new Date();
```

Date 类的实例表示一个特定时间，可以查询或修改该时间的属性（如月、日、小时和秒）。Date 类用于检索相对于通用时间（格林尼治时间，现称为通用时间或 UTC）或相对于本地时间（由运行 Flash Player 的操作系统上的本地时区设置决定）的日期和时间值。

4. Error

Error 数据类型是由 Error 类定义的，该类包含有关脚本中出现的错误的信息。开发 ActionScript 3.0 应用程序过程中，如果在 Flash Player 的调试版中运行已编译的代码，将弹出对话框显示 Error 类型的异常或子类异常，以帮助开发者排除代码中的故障。可以使用 Error 构造函数来创建 Error 对象。通常，将新的 Error 对象从 try 代码块中引发，然后由 catch 或 finally 代码块捕获。

```
try
{
   //可能会引发错误的一些代码
}
catch (err:Error)
{
   //用于响应错误的代码
}
```

```
finally
{
   //无论是否引发错误都会运行的代码。此代码可在发生错误之后清除错误，或者采取措施使应用程
   //序继续运行。
}
```


5. Function

函数是可在 ActionScript 中调用的基本代码单位。ActionScript 中用户定义的函数和内置函数都由 Function 对象来表示，该对象是 Function 类的实例。

类的方法与 Function 对象略有不同。与普通函数对象不同，类的方法和与其关联的类对象紧密联系。因此，方法或属性具有在同一类的所有实例中共同的定义。可以从实例中提取方法并将其处理为“绑定”方法（保留与原始实例的链接）。对于绑定方法，this 关键字指向实现该方法的原始对象。对于函数，this 在调用函数时指向关联对象。

6. RegExp

RegExp 数据类型是由 RegExp 类定义的，该类允许使用正则表达式（即可用于在字符串中执行搜索和替换文本的模式）。

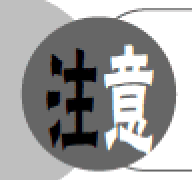
可以使用 new RegExp()构造函数或将 RegExp 文本分配给一个变量，从而创建新的 RegExp 对象，如下所示。

```
var pattern1:RegExp = new RegExp("test-\\d", "i");
var pattern2:RegExp = /test-\d/i;
```

7. XML

XML 数据类型是由 XML 类定义的，该类包含用于处理 XML 对象的方法和属性。XML 类（以及 XMLList、Namespace 和 QName 类）可实现 ECMAScript for XML（E4X）规范（ECMA—357 第 2 版）中定义的强大的 XML 处理标准。

使用 toXMLString()方法可返回 XML 对象的字符串表示形式，不管该 XML 对象具有简单内容还是复杂内容。



ActionScript 2.0 中的 XML 类以及相关类已重命名为 XMLDocument 并移到 flash.xml 包中，包含在 ActionScript 3.0 中，以实现向后兼容。

8. XMLList

XMLList 数据类型是由 XMLList 类定义的，该类中包含用于处理一个或多个 XML 元素的方法。XMLList 对象可以表示一个或多个 XML 对象或元素（包括多个节点或属性），因此，可以对作为一个组的多个元素调用方法，也可以对集合中的各个元素分别调用方法。

如果 XMLList 对象只包含一个 XML 元素，那么可以直接对 XMLList 对象使用 XML 类方法。在下面的实例中，example 为长度 1 的 XMLList 对象，因此可以对它调用任意 XML 方法。

```
var example=<example><two>2</two></example>;
```

如果试图对包含多个 XML 元素的 XMLList 对象使用 XML 类方法，则会引发异常。此时，应遍历 XMLList 集合（例如，使用 for each…in 语句），并对集合中的每个 XML 对象应用方法。

3.2.3 数据类型检查

类型检查可以在编译时或运行时执行。静态类型语言（如 C++和 Java），在编译时执行类型检查。动态类型语言（如 Smalltalk 和 Python），在运行时执行类型检查。ActionScript 3.0 是动态类型语言，它在运行时执行类型检查，同时也支持在名为“严格模式”的特殊编译器模式下，编译时执行类型检查。在严格模式下，类型检查既发生在编译时，也发生在运行时，但是在标准模式下，类型检查仅发生在运行时。

在构造代码时，动态类型语言带来了极大的灵活性，但代价是在运行时可能出现类型错误。静态类型语言在编译时报告类型错误，但代价是要求类型信息在编译时是已知的。

1. 编译时类型检查

在较大的项目中通常建议使用编译时类型检查，因为随着项目变大，能够尽早捕获类型错误，对于程序的稳定性相当重要。这也是为什么将 Adobe Flash CS3 Professional 和 Adobe Flex Builder 3 中的 ActionScript 编译器默认设置为在严格模式下运行。

为了提供编译时类型检查，编译器需要知道代码中的变量或表达式的数据类型信息。为了显式声明变量的数据类型，在变量名后面添加冒号运算符（:）作为其后缀。要将数据类型与参数相关联，也应使用冒号运算符。例如，下面的代码向 xParam 参数中添加数据类型信息，并用显式数据类型声明变量 myParam。

```
internal function runtimeTest(xParam:String):void{
    trace(xParam);
}
internal function initApp():void{
    var myParam:String = "hello";
    runtimeTest(myParam);
}
```

在严格模式下，ActionScript 编译器将类型不匹配报告为编译器错误。例如，下面的代码声明一个 Object 类型的函数参数 xParam，但是之后又尝试向该参数赋予 String 类型和 Number 类型的值。这在严格模式下会产生编译器错误。

```
internal function dynamicTest(xParam:Object):void{
    if (xParam is String)
    {
        var myStr:String = xParam; // 在严格模式下产生编译器错误
        trace("String: " + myStr);
    }
    else if (xParam is Number)
```

```
        {
            var myNum:Number = xParam; // 在严格模式下产生编译器错误
            trace("Number: " + myNum);
        }
    }
```

但是，即使在严格模式下，也可以选择不在赋值语句右侧指定类型，从而避开编译时类型检查。可以通过省略类型注释或使用特殊的符号（*）类型注释，来将变量或表达式标记为无类型。例如修改上面的代码后的代码如下所示。

```
internal function dynamicTest(xParam):void
{
    if (xParam is String)
    {
        var myStr:String = xParam;
        trace("String: " + myStr);
    }
    else if (xParam is Number)
    {
        var myNum:Number = xParam;
        trace("Number: " + myNum);
    }
}
internal function initApp():void{
    dynamicTest(100)
    dynamicTest("one hundred");
}
```

2. 运行时类型检查

在 ActionScript 3.0 中，无论是在严格模式下还是在标准模式下编译，在运行时都将执行类型检查。考虑以下情形：将值 3 作为一个参数传递给需要数组的函数。在严格模式下，编译器将生成一个错误，因为值 3 与 Array 数据类型不兼容。如果禁用了严格模式而在标准模式下运行，则编译器将不会指出类型不匹配，但是当 Flash Player 在运行时执行类型检查时则会产生运行时错误。

下面的示例说明一个名为 typeTest()的函数，该函数需要一个 Array 参数，但是接收到的是值 3。在标准模式下这会产生运行时错误，因为值 3 不是参数声明的数据类型（Array）的变量。

```
function typeTest(xParam:Array)
{
    trace(xParam);
}
var myNum:Number = 3;
//在 ActionScript 3.0 标准模式下出现运行时错误
```

```
typeTest(myNum);
```

在严格模式下，使用无类型变量时，虽然避开了编译时类型检查，但不会消除类型检查，而只是将其延迟到运行时执行。例如，如果上一个示例中的 myNum 变量没有声明数据类型，那么，编译器将检测不到类型不匹配，但是 Flash Player 将生成一个运行时错误，因为它会将 myNum 运行时的值（赋值语句将其设置为 3）与 xParam 的类型（设置为 Array 数据类型）进行比较。


```
function typeTest(xParam:Array)
{
    trace(xParam);
}
var myNum = 3;
// 在 ActionScript 3.0 严格模式下出现运行时错误
typeTest(myNum);
```

与编译时类型检查相比，运行时类型检查还允许更灵活地使用继承。标准模式会将类型检查延迟到运行时执行，从而允许引用子类的属性。当使用父类来声明变量，并且使用子类来实例化时，就只能在标准模式下运行。例如，可以创建一个名为 ClassBase 的可扩展类（具有 final 属性的类不能扩展），如下所示。

```
class ClassBase
{0
}
```

随后，创建一个名为 ClassExtender 的 ClassBase 子类，该子类具有一个名为 someString 的属性，如下所示。

```
class ClassExtender extends ClassBase
{
    var someString:String;
}
```

通过使用这两个类，可以创建一个使用 ClassBase 类型进行声明，但使用 ClassExtender() 构造函数进行实例化的变量。这是因为父类 ClassBase 不包含子类 ClassExtender 的任何属性和方法。

```
var myClass:ClassBase = new ClassExtender();
```

但是，子类中则包含了其父类中没有的属性或方法。例如，ClassExtender 类中包含 someString 属性，该属性在 ClassBase 类中不存在。在 ActionScript 3.0 标准模式下，可以使用 myClass 实例来引用此属性，而不会生成编译时错误，如下面的代码所示。

```
var myClass:ClassBase = new ClassExtender();
//在 ActionScript 3.0 标准模式下不出现错误
myClass.someString = "hello";
```

3.2.4 is 和 as 运算符

上一节介绍了编译器和 Flash Player 对数据类型的检查，本节介绍 is 和 as 运算符，它们可以实现用户对数据类型的检查。instanceof 运算符在 ActionScript 3.0 之前的版本中就有，只是功能不够强大。下面详细介绍 is 和 as 运算符，同时和 instanceof 运算符做区别分析。

1. is 运算符

is 运算符是 ActionScript 3.0 中的新增运算符，它可用来测试变量或表达式是否为给定数据类型的成员。在早期的 ActionScript 版本中，此功能由 instanceof 运算符提供。但在 ActionScript 3.0 中，不应使用 instanceof 运算符来测试变量或表达式是否为给定数据类型的成员。对于程序中的类型检查，应用 is 运算符来代替 instanceof 运算符，因为表达式 x instanceof y 只是在 x 的原型链中检查 y 是否存在（在 ActionScript 3.0 中，原型链不能全面地描述继承层次结构）。

is 运算符检查正确的继承层次结构，不但可以用来检查对象是否为特定类的实例，而且还可以检查对象是否是用来实现特定接口的类的实例。下面的示例创建 Sprite 类的一个名为 mySprite 的实例，并使用 is 运算符来测试 mySprite 是否为 Sprite 和 DisplayObject 类的实例，以及它是否实现 IEventDispatcher 接口。

```
var mySprite:Sprite = new Sprite();
trace(mySprite is Sprite);                // true
trace(mySprite is DisplayObject);         // true
trace(mySprite is IEventDispatcher);      // true
```

is 运算符检查继承层次结构，并正确地报告 mySprite 与 Sprite 和 DisplayObject 类兼容（Sprite 类是 DisplayObject 类的子类）。is 运算符还检查 mySprite 是否是从实现 IEventDispatcher 接口的任意类继承的。由于 Sprite 类是从实现 IEventDispatcher 接口的 EventDispatcher 类继承的，因此 is 运算符会正确地报告 mySprite 也实现该接口。

下面的示例进行与上一个示例相同的测试，但使用的是 instanceof 运算符，而不是 is 运算符。instanceof 运算符正确地识别出 mySprite 是 Sprite 或 DisplayObject 的实例，但是，当它用来测试 mySprite 是否实现 IEventDispatcher 接口时，返回的却是 false。

```
trace(mySprite instanceof Sprite);            // true
trace(mySprite instanceof DisplayObject);     // true
trace(mySprite instanceof IEventDispatcher);  // false
```

2. as 运算符

as 运算符是 ActionScript 3.0 中的新增运算符，也可用来检查表达式是否为给定数据类型的成员。但是，与 is 运算符不同的是，as 运算符不返回布尔值，而是返回表达式的值（代替 true）或 null（代替 false）。下面的例子说明了在简单情况下使用 as 运算符替代 is 运算符的结果，例如，检查 Sprite 实例是否为 DisplayObject、IEventDispatcher 和 Number 数据类型的

成员。

```
var mySprite:Sprite = new Sprite();
trace(mySprite as Sprite);                // [Sprite 对象]
trace(mySprite as DisplayObject);         // [Sprite 对象]
trace(mySprite as IEventDispatcher);      // [Sprite 对象]
trace(mySprite as Number);                // null
```

在使用 as 运算符时，右侧的操作数必须是数据类型。如果尝试使用表达式（而非数据类型）作为右侧的操作数，将会产生错误。

3.2.5 数据类型转换

在将某个值转换为其他数据类型的值时，就说发生了类型转换。类型转换可以是“隐式”的，也可以是“显式”的。隐式转换又称为“强制转换”。例如，如果将值 2 赋给 Boolean 数据类型的变量，则 Flash Player 会先将值 2 转换为布尔值 true，然后再将其赋给该变量。显式转换，是在代码指示编译器将一个数据类型的变量，视为属于另一个数据类型时发生。在涉及基本数据类型的值时，转换功能将一个数据类型的值实际转换为另一个数据类型的值。要将对象转换为另一类型，用小括号括起对象名，并在它前面加上新类型的名称。例如，下面的代码提取一个布尔值并将它转换为一个整数。

```
var myBoolean:Boolean = true;
var myINT:int = int(myBoolean);
trace(myINT); // 1
```

1. 隐式转换

在运行时，会经常发生隐式转换，如下列出了常见情况。

在赋值语句中。

在将值作为函数的参数传递时。

在从函数中返回值时。

在使用某些运算符（如加法运算符（+））的表达式中。

对于用户定义的类型，当要转换的值是目标类（或者派生自目标类的类）的实例时，隐式转换会成功。如果隐式转换不成功，就会出现错误。例如，下面的代码中包含成功的隐式转换和不成功的隐式转换。

```
class A {}
class B extends A {}
var objA:A = new A();
var objB:B = new B();
var arr:Array = new Array();
objA = objB;     // 转换成功
objB = arr;      // 转换失败
```

对于基本数据类型而言，隐式转换是通过调用内部转换算法来处理的，该算法与显式转

换函数所调用的算法相同。下面会详细讲述这些基本数据类型的转换。

2. 显式转换

在严格模式下进行编译时，使用显式转换会非常有用，因为有时会不希望因类型不匹配而生成编译时错误。当知道强制转换功能会在运行时正确转换变量的值时，就可以使用显式转换。例如，在处理从表单接收的数据时，依赖强制转换功能将某些字符串值转换为数值。下面的代码会生成编译时错误，即使代码在标准模式下能够正确运行。

```
var quantityField:String = "3";
var quantity:int = quantityField; // 在严格模式下出现编译时错误
```

如果继续使用严格模式，但是希望将字符串转换为整数，则可以使用显式转换，如下所示：

```
var quantityField:String = "3";
var quantity:int = int(quantityField); // 显式转换成功
```

3. 转换为 int、uint 和 Number

可以将任何数据类型转换为以下 3 种数字类型：int、uint 和 Number。如果 Flash Player 由于某种原因而无法转换数字，则会为 int 和 uint 数据类型赋予默认值 0，为 Number 数据类型赋予默认值 NaN。如果将布尔值转换为数字，则 true 变成值 1，false 变成值 0，从如下所示的代码可以看到效果。

```
var myBoolean:Boolean = true;
var myUINT:uint = uint(myBoolean);
var myINT:int = int(myBoolean);
var myNum:Number = Number(myBoolean);
trace(myUINT, myINT, myNum); // 1 1 1
myBoolean = false;
myUINT = uint(myBoolean);
myINT = int(myBoolean);
myNum = Number(myBoolean);
trace(myUINT, myINT, myNum); // 0 0 0
```

仅包含数字的字符串值可以成功地转换为数字类型之一。看上去像负数的字符串或者表示十六进制值的字符串（例如，0x1A）也可以转换为数字类型。转换过程中会忽略字符串值中的前导或尾随空白字符。还可以使用 Number()方法来转换看上去像浮点数的字符串。如果包含小数点，则用 uint()方法和 int()方法转换，返回一个整数，小数点和它后面的字符被截断。例如，下面的字符串值可以转换为数字。

```
trace(uint("5"));        // 5
trace(uint("-5"));       // 4294967291
trace(uint(" 27 "));     // 27
trace(uint("3.7"));      // 3
trace(int("3.7"));       // 3
```

```
trace(int("0x1A"));     // 26
trace(Number("3.7"));   // 3.7
```

对于包含非数字字符的字符串值，在用 int()或 uint()转换时，将返回 0；在用 Number()转换时，将返回 NaN。转换过程中会忽略前导和尾随空白，但是，如果字符串中包含将两个数字隔开的空白，则将返回 0 或 NaN。

```
trace(uint("5a"));      // 0
trace(uint("ten"));     // 0
trace(uint("17 63"));   // 0
```

在 ActionScript 3.0 中，Number()函数不再支持八进制数或基数为 8 的数。如果向 ActionScript 2.0 中的 Number()函数，提供的字符串中包含前导 0，则该数字将被视为八进制数，并将转换为等效的十进制数。对于 ActionScript 3.0 中的 Number()函数，情况并非如此，因为 ActionScript 3.0 会忽略前导 0。例如，在使用不同的 ActionScript 版本进行编译时，下面的代码会生成不同的输出结果。

```
trace(Number("044"));
// ActionScript 3.0    44
// ActionScript 2.0    36
```

将一种数值类型的值赋给另一种数值类型的变量时，转换并不是必须的。即使在严格模式下，数值类型也会隐式转换为其他数值类型。这意味着，在某些情况下，在超出类型的范围时，可能会生成意外的值。下面的几个例子全部是在严格模式下进行编译的，但是某些示例将生成意外的值。

```
internal function initApp():void{
    var testint=-3;
    var myUInt:uint = testint;          // 将 int/Number 值赋给 uint 变量
    trace(myUInt); // 4294967293
    var myNum:Number = myUInt;          // 将 int/uint 值赋给 Number 变量
    trace(myNum) // 4294967293
    var myInt:int = uint.MAX_VALUE + 1; // 将 Number 值赋给 uint 变量
    trace(myInt); // 0
    myInt = int.MAX_VALUE + 1;          // 将 uint/Number 值赋给 int 变量
    trace(myInt); // -2147483648
}
```

表 3-3 概述了将其他数据类型转换为 Number、int 或 uint 数据类型的结果。

表 3-3　转换为 Number、int 或 uint 数据类型的结果

数据类型或值	转换为 Number、int 或 uint 的结果
Boolean	如果值为 true，则结果为 1；否则为 0
Date	Date 对象的内部表示形式，即从 1970 年 1 月 1 日午夜（通用时间）以来所经过的毫秒数
null	0
Object	如果实例为 null 并转换为 Number，则结果为 NaN；否则为 0

续表

数据类型或值	转换为 Number、int 或 uint 的结果
String	如果 Flash Player 可以将字符串转换为数字，则结果为数字；否则，如果转换为 Number，则结果为 NaN，如果转换为 int 或 uint，则结果为 0
undefined	如果转换为 Number，则结果为 NaN；如果转换为 int 或 uint，则结果为 0

4．转换为 Boolean

在从任何数值数据类型（uint、int 和 Number）转换为 Boolean 时，如果数值为 0，则结果为 false，否则为 true。对于 Number 数据类型，如果值为 NaN，则结果也为 false。下面的例子说明在转换–1、0 和 1 等数字时的结果。

```
var myNum:Number;
for (myNum = -1; myNum<2; myNum++)
{
    trace("Boolean(" + myNum +") is " + Boolean(myNum));
}
```

上述代码的输出结果如下所示，说明在这 3 个数字中，只有 0 返回 false 值。

```
Boolean(-1) is true
Boolean(0) is false
Boolean(1) is true
```

在将字符串值转换为 Boolean 数据类型时，如果字符串为 null 或空字符串（""），则会返回 false，否则将返回 true，如下所示的代码将字符串转换为 Boolean 数据类型。

```
var str1:String;          // 未初始化的字符串为 null。
trace(Boolean(str1));     // false
var str2:String = "";     // 空字符串
trace(Boolean(str2));     // false
var str3:String = " ";    // 仅空白
trace(Boolean(str3));     // true
```

在将 Object 类的实例转换为 Boolean 数据类型时，如果该实例为 null，则将返回 false；否则将返回 true，如下所示的代码将 Object 实例转换为 Boolean 数据类型。

```
var myObj:Object;         // 未初始化的对象为 null。
trace(Boolean(myObj));    // false
myObj = new Object();     // 实例化
trace(Boolean(myObj));    // true
```

在严格模式下，系统会对布尔变量进行特殊处理，因为不必转换即可向布尔变量赋予任何数据类型的值。即使在严格模式下，也可以将所有的数据类型隐式转换为 Boolean 数据类型。换言之，与几乎所有其他的数据类型不同，转换为 Boolean 数据类型不是避免在严格模式下出错所必须的。下面的几个例子全部是在严格模式下编译的，它们在运行时按照预期的方式工作。

```
var myObj:Object = new Object();    // 实例化
var bool:Boolean = myObj;
trace(bool);                        // true
bool = "random string";
trace(bool);                        // true
bool = new Array();
trace(bool);                        // true
bool = NaN;
trace(bool);                        // false
```

表 3-4 概述了其他数据类型转换为 Boolean 数据类型时的结果。

表 3-4　转换为 Boolean 数据类型的结果

数据类型或值	转换为 Boolean 数据类型的结果
String	如果值为 null 或空字符串（""），则结果为 false；否则为 true
null	False
Number、int 或 uint	如果值为 NaN 或 0，则结果为 false；否则为 true
Object	如果实例为 null，则结果为 false；否则为 true

5．转换为 String

从任何数值数据类型转换为 String 数据类型时，都会返回数字的字符串表示形式。在将布尔值转换为 String 数据类型时，如果值为 true，则返回字符串 true；如果值为 false，则返回字符串 false。

在从 Object 类的实例转换为 String 数据类型时，如果该实例为 null，则返回字符串 null。否则，将返回字符串[object Object]。

在从 Array 类的实例转换为 String 数据类型时，会返回一个字符串，其中包含所有数组元素的逗号分隔列表。例如，下面的例子在转换为 String 数据类型时，将返回一个包含数组中全部元素的字符串。

```
var myArray:Array = ["primary", "secondary", "tertiary"];
trace(String(myArray));     // primary,secondary,tertiary
```

在从 Date 类的实例转换为 String 数据类例时，会返回该实例所包含日期的字符串表示形式。例如，下面的例子返回 Date 类实例的字符串表示形式（输出结果显示的是太平洋夏令时）。

```
var myDate:Date = new Date(2005,6,1);
trace(String(myDate));     // 星期五 7 月 1 日 00:00:00 GMT-0700 2005
```

表 3-5 概述了将其他数据类型转换为 String 数据类型时的结果。

表 3-5　转换为 String 数据类型的结果

数据类型或值	转换为 String 数据类型的结果
Array	一个包含所有数组元素的字符串
Boolean	“true”或“false”
Date	Date 对象的字符串表示形式
null	“null”
Number、int 或 uint	数字的字符串表示形式
Object	如果实例为 null，则结果为“null”；否则为“[object Object]”

3.3　运算符

运算符是一种特殊的函数，它们具有一个或多个操作数并返回相应的值。操作数是被运算符用作输入的值，通常是字面值、变量或表达式。例如，在下面的代码中，将加法运算符（+）和乘法运算符（*）与 3 个字面值操作数（2、3 和 4）结合使用来返回一个值。赋值运算符（=），随后使用该值所返回的值 14 赋给变量 sumNumber。

```
var sumNumber:uint = 2 + 3 * 4; // uint = 14
```

有些运算符是重载的，这意味着它们的行为因传递给它们的操作数的类型或数量而异。例如，加法运算符（+）就是一个重载运算符，其行为因操作数的数据类型而异。如果两个操作数都是数字，则加法运算符会返回这些值的和。如果两个操作数都是字符串，则加法运算符会返回这两个操作数连接后的结果。下面的例子代码说明运算符的行为如何因操作数而异。

```
trace(5 + 5);        // 10
trace("5" + "5");    // 55
```

运算符的行为还可能因所提供的操作数的数量而异。减法运算符（–）既是一元运算符又是二元运算符。对于减法运算符，如果只提供一个操作数，则该运算符会对操作数求反并返回结果；如果提供两个操作数，则减法运算符返回这两个操作数的差。下面的例子说明首先将减法运算符用作一元运算符，然后再将其用作二元运算符。

```
trace(-3);           // -3
trace(7-2);          // 5
```

3.3.1　运算符的分类

运算符按照操作数的多少，可以分为一元、二元和三元运算符。其中一元运算符有 1 个操作数。例如，递增运算符（++）就是一元运算符，因为它只能有一个操作数。二元运算符有两个操作数。例如，除法运算符（/）有两个操作数。三元运算符有 3 个操作数。例如，条件运算符（?:）具有 3 个操作数。

根据运算符的作用可将运算符划分为多种类型，常用的运算符有主要运算符、后缀运算符、一元运算符、乘法运算符、加法运算符等。具体如表 3-6 所示。

表 3-6　运算符及其类型

类别	运算符
主要运算符	[] {x:y} () f(x) new x.y x[y] <></> @ :: ..
后缀运算符	x++ x– –
一元运算符	++x – –x + – ~ ! delete typeof void
乘法运算符	* / %
加法运算符	+ –

续表

<table>
<tr><th>类别</th><th>运算符</th></tr>
<tr><td>按位移位运算符</td><td><< >> >>></td></tr>
<tr><td>关系运算符</td><td>< > <= >= as in instanceof is == != === !==</td></tr>
<tr><td>逻辑运算符</td><td>& | ^ ~ && || !</td></tr>
<tr><td>条件运算符</td><td>?:</td></tr>
<tr><td>赋值运算符</td><td>= += –= *= /= %= &= |= ^= <<= >>= >>>=</td></tr>
</table>

3.3.2 常用运算符

下面根据运算符的作用进行分类，详细介绍常用运算符的作用及执行的运算。

1．算术运算符

算术运算符，也就是将变量进行算术运算，例如加、减、乘、除等。ActionScript 3.0 中提供的算术运算符如表 3-7 所示。

表 3-7 算术运算符及执行的运算

运算符	执行的运算	运算符	执行的运算
*	乘法	+	加法
/	除法	–	减法
%	求模		

2．逻辑运算符

逻辑运算符有两个操作数，它返回布尔型结果。表 3-8 中列出了所有的逻辑运算符。

表 3-8 逻辑运算符及执行的运算

<table>
<tr><th>运算符</th><th>执行的运算</th><th>运算符</th><th>执行的运算</th></tr>
<tr><td>&</td><td>按位“与”</td><td>>=</td><td>大于或等于</td></tr>
<tr><td>^</td><td>按位“异或”</td><td>&&</td><td>逻辑“与”</td></tr>
<tr><td>|</td><td>按位“或”</td><td>||</td><td>逻辑“或”</td></tr>
</table>

3．赋值运算符

最常见的赋值运算符是“=”，该运算符将右操作数的值存储在左操作数表示的存储位置、属性或索引器中，并将值作为结果返回。操作数的类型必须相同（或右操作数必须可以隐式转换为左操作数的类型）。表 3-9 中列出了所有的逻辑运算符。

表 3-9 赋值运算符及执行的运算

<table>
<tr><th>运算符</th><th>执行的运算</th><th>运算符</th><th>执行的运算</th></tr>
<tr><td>=</td><td>赋值</td><td><<=</td><td>按位向左移位赋值</td></tr>
<tr><td>*=</td><td>乘法赋值</td><td>>>=</td><td>按位向右移位赋值</td></tr>
<tr><td>/=</td><td>除法赋值</td><td>>>>=</td><td>按位无符号向右移位赋值</td></tr>
<tr><td>%=</td><td>求模赋值</td><td>&=</td><td>按位“与”赋值</td></tr>
<tr><td>+=</td><td>加法赋值</td><td>^=</td><td>按位“异或”赋值</td></tr>
<tr><td>-=</td><td>减法赋值</td><td>|=</td><td>按位“或”赋值</td></tr>
</table>

基本格式如下所示。

```
int x=3;      //程序将把赋值运算符右边的值赋值给左边的变量
int a=x=1;    //程序先将 1 赋值给已经声明的 int 类型的变量 x，再将变量 x 的值赋值给 int 类型
              //的变量 a
```

4．后缀运算符

后缀运算符只有一个操作数，它递增或递减该操作数的值。虽然这些运算符是一元运算符，但是它们有别于其他一元运算符，被单独划归到了一个类别，因为它们具有更高的优先级和特殊的行为。在将后缀运算符用作较长表达式的一部分时，会在处理后缀运算符之前返回表达式的值。表 3-10 列出了所有的后缀运算符，下面的代码说明如何在递增值之前返回表达式 xNum++的值。

```
var xNum:Number = 0;
trace(xNum++);  // 0
trace(xNum);    // 1
```

表 3-10　后缀运算符及执行的运算

运算符	执行的运算	运算符	执行的运算
++	递增（后缀）	--	递减（后缀）

5．一元运算符

一元运算符只有一个操作数。这一组中的递增运算符（++）和递减运算符（--）是前缀运算符，这意味着它们在表达式中出现在操作数的前面。前缀运算符与它们对应的后缀运算符不同，因为递增或递减操作是在返回整个表达式的值之前完成的。表 3-11 列出了所有的一元运算符，下面的代码说明如何在递增值之后返回表达式++xNum 的值。

```
var xNum:Number = 0;
trace(++xNum); // 1
trace(xNum);   // 1
```

表 3-11　一元运算符及执行的运算

运算符	执行的运算	运算符	执行的运算
++	递增（前缀）	~	按位“非”
--	递减（前缀）	delete	删除属性
+	一元+	typeof	返回类型信息
-	一元-（非）	void	返回 undefined 值
!	逻辑“非”		

6．按位移位运算符

按位移位运算符有两个操作数，它将第一个操作数的各位按第二个操作数指定的长度移位。表 3-12 中列出了所有的按位移位运算符。

表 3-12　按位移位运算符及执行的运算

运算符	执行的运算	运算符	执行的运算
<<	按位向左移位	>>>	按位无符号向右移位
>>	按位向右移位		

7．关系运算符

关系运算符有两个操作数，它比较两个操作数的值，然后返回一个布尔值。表 3-13 中列出了所有的关系运算符。

表 3-13　关系运算符及执行的运算

运算符	执行的运算	运算符	执行的运算
<	小于	instanceof	检查原型链
>	大于	is	检查数据类型
<=	小于或等于	==	等于
>=	大于或等于	!=	不等于
as	检查数据类型	===	严格等于
in	检查对象属性	!==	严格不等于

8．条件运算符

条件运算符（?:）是 if…else 结构的简化形式，使用这种结构能够使程序更简洁、雅观地表达那些简单的 if…else 结构。条件运算符是唯一的三元运算符，之所以称为三元是因为它带有 3 个操作数。在编程过程中，使用该运算符可以计算一个条件，如果条件为真，就返回一个值；如果条件为假，则返回另一个值。其语法如下所示。

```
condition?true_value:false_value
```

其中，condition 是要计算的 Boolean 型表达式，true_value 是 condition 为 true 时返回的值，false_value 是 condition 为 false 时返回的值。

9．其他运算符

其他运算符包括那些用来创建 Array 和 Object 字面值、对表达式进行分组、调用函数、实例化类以及访问属性的运算符。表 3-14 中列出了所有的主要运算符。

表 3-14　主要运算符及执行的运算

运算符	执行的运算	运算符	执行的运算
[]	初始化数组	x.y x[y]	访问属性
{x:y}	初始化对象	<></>	初始化 XMLList 对象（E4X）
()	对表达式进行分组	@	访问属性（E4X）
f(x)	调用函数	::	限定名称（E4X）
new	调用构造函数	..	访问子级 XML 元素（E4X）

3.3.3　运算符的优先级

在上一节中分别学习各种类型的运算符，那么如果在同一条语句上使用多个运算符，程

序会先运行哪个呢？这就产生了运算符之间优先级的问题，先看一个小例子。

```
test=6+7*2;
```

得到的结果不是 26，而是 20，因为程序先计算了 7*2 然后再与 6 相加。幸运的是，可以通过用括号的优先运算来改变计算的顺序。这种程序运算符先后的计算顺序就称为运算符的优先级，表 3-15 按优先级递减的顺序列出了 ActionScript 3.0 中的运算符。该表内同一行中的运算符具有相同的优先级，每行运算符都比位于其下方的运算符的优先级高。

表 3-15　运算符之间的运算优先级

<table>
<tr><th>优先级</th><th>类别</th><th>运算符</th></tr>
<tr><td>1</td><td>主要运算符</td><td>[] {x:y} () f(x) new x.y x[y] <></> @ :: ..</td></tr>
<tr><td>2</td><td>后缀运算符</td><td>x++ x– –</td></tr>
<tr><td>3</td><td>一元运算符</td><td>++x – –x + – ~ ! delete typeof void</td></tr>
<tr><td>4</td><td>乘法运算符</td><td>* / %</td></tr>
<tr><td>5</td><td>加法运算符</td><td>+ –</td></tr>
<tr><td>6</td><td>按位移位运算符</td><td><< >> >>></td></tr>
<tr><td>7</td><td>关系运算符</td><td>< > <= >= as in instanceof is == != === !==</td></tr>
<tr><td>8</td><td>逻辑运算符</td><td>& | ^ ~ && || !</td></tr>
<tr><td>9</td><td>条件运算符</td><td>?:</td></tr>
<tr><td>10</td><td>赋值运算符</td><td>= += –= *= /= %= &= |= ^= <<= >>= >>>=</td></tr>
</table>

3.4　流程控制语句

ActionScript 3.0 作为一种脚本语言，也有其自己的流程控制语句。在掌握了 ActionScript 3.0 程序的基本运算符等基本语法后，本节将主要介绍与 ActionScript 3.0 有关的流程控制语句，包括条件语句和循环语句。

3.4.1　条件语句

使用条件语句可以对预先设定的条件进行判断。同时，这样的条件语句一般都会结合一个语句块或者普通语句。当条件语句对预设的条件的逻辑判断结束以后就转到附带的语句块或者语句去执行相应的操作。ActionScript 3.0 提供了多种条件语句。

if…else 语句

switch…case 语句

if…else 语句用于判断条件是否为 true 或者 false，并且根据计算结果指定要执行的操作。通常情况下，判断条件是用比较运算符对数值或者变量进行比较运算的表达式。

在判断条件为 true 时运行指定的语句块，则在判断条件为 false 时跳过指定的语句块。下面的例子演示了这种情况下的 if…else 语句。注意此例省略了关键字 else。

```
var a:int=9;
if(a<10)
```

```
{
    trace(a+"是 1 位数!");
}
```

可以使用 if…else 语句定义两个可执行语句块：判断条件为 true 时运行某一语句块，条件为 false 时运行另一语句块。例如，下面的例子就演示这种情况下的 if…else 语句。

```
var i:int=9,j:int =8;
if(i>j)
{
    trace("变量 i 的值大于变量 j 的值!");
}
else
{
    trace("变量 i 的值小于变量 j 的值!");
}
```

除此之外，if…else 语句的另一种变形允许从多个条件中选择，即添加 else if 子句以扩充 if…else 语句的功能，使得可以控制基于多种可能的程序流程，如下所示。

```
if (x > 20)
{
   trace("x is > 20");
}
else if (x < 0)
{
   trace("x 是个负值");
}
```

在 if…else 语句中，可以添加任意多个 else if 子句以提供多种选择，但是使用多个 else if 子句会使代码变得非常复杂。在多个条件中进行选择的更好方法是使用 switch…case 语句。

switch…case 语句提供了 if…else 语句的一个变通形式，可以从多个语句块中选择其中的一个执行。switch…case 语句提供的功能与 if…else 语句类似，但是可以使代码更加简练、易读。switch…case 语句在其开始处使用一个简单的测试表达式。表达式的结果将与结构中每个 case 子句的值进行比较。如果匹配，则执行与该 case 关联的语句块。语句块以 case 语句开头，以 break 语句结尾。

下面使用 switch…case 语句，根据 Date.getDay()方法返回的日期值，输出对应的星期，如代码 3.1 所示。

代码 3.1　使用 switch…case 语句

```
var someDate:Date = new Date();
var dayNum:uint = someDate.getDay();
switch(dayNum)
{
   case 0:
```

```
        trace("星期天");
        break;
    case 1:
        trace("星期一");
        break;
    case 2:
        trace("星期二");
        break;
    case 3:
        trace("星期三");
        break;
    case 4:
        trace("星期四");
        break;
    case 5:
        trace("星期五");
        break;
    case 6:
        trace("星期六");
        break;
    default:
        trace("超出范围");
        break;
}
```

需要注意 switch…case 语句只计算一次开始处的一个表达式，而 if…else 语句计算每个 else if 子句的表达式，这些表达式可以各不相同。仅当每个 else if 子句计算的表达式都相同时，才可以使用 switch…case 语句代替 if…else 语句。

3.4.2　循环语句

循环语句也称为迭代语句，使用循环语句可以反复执行某个代码块，直到循环结束条件满足后才停止执行。ActionScript 3.0 提供 5 种不同的循环机制，如 for、for…in、while、do…while 和 for each…in。下面详细介绍这几种循环语句。

1. for 循环

for 循环语句在程序执行前测试是否满足某个条件。其语法如下：

```
for(初始值表达式;循环条件表达式;循环后的操作表达式)
{
     执行语句块
}
```

初始值表达式在程序刚进入 for 循环的时候第一时间被执行，当程序执行完初始值表达式

以后，程序会继续执行循环条件表达式，循环条件是一条判断语句，如果循环条件表达式返回的结果为 true，程序就会执行大括号里的执行语句块的语句。直到循环条件表达式返回的结果为 false 时程序才会跳出这整个 for 循环。下面以例子说明 for 循环语句，如代码 3.2 所示。

代码 3.2　for 循环

```
var i:int;
for (i = 0; i < 5; i++)
{
    trace(i);
}
```

上述代码的输出结果如下：

```
0
1
2
3
4
```

2. for…in 循环

for…in 循环用于循环访问对象属性或数组元素。例如，可以使用 for…in 循环来循环访问通用对象的属性（不按任何特定的顺序来访问对象的属性，因此属性可能以看似随机的顺序出现），代码 3.3 所示为使用 for…in 语句访问对象属性、输出属性的名称及值的例子。

代码 3.3　使用 for…in 循环输出对象属性

```
var myObj:Object = {x:20, y:30};
for (var i:String in myObj)
{
    trace(i + ": " + myObj[i]);
}
```

上述代码的输出结果如下：

```
x: 20
y: 30
```

如果对象是自定义类的一个实例，则除非该类是动态类，否则将无法循环访问该对象的属性。即便是动态类的实例，也只能循环访问动态添加的属性。

使用 for…in 循环，还可以访问数组中的元素，代码 3.4 创建了一个数组，利用 for…in 循环语句输出数组中的元素。

代码 3.4　使用 for…in 循环输出数组中的元素

```
var myArray:Array = ["one", "two", "three"];
for (var i:String in myArray)
```

```
{
    trace(myArray[i]);
}
```

上述代码的输出结果如下：

```
one
two
three
```


3．while 循环

与 for 循环一样，while 也是一个预测试的循环。while 循环通常用于下述情况，在循环开始之前不知道重复执行一个语句或语句块的次数。其语法如下：

```
while(条件表达式)
{
     //执行的语句块
}
```

条件表达式也是一个布尔表达式，控制内含语句被执行的次数。可以使用 break 或 continue 语句来控制 while 语句中的执行语句，运行方式和 for 语句中的完全相同。

```
var i:int = 0;
while (i < 5)
{
    trace(i);
    i++;
}
```



所有 ActionScript 3.0 循环中，如果只执行一条语句，而不是语句块，可以省略花括号，但是最好在任何情况下都加上花括号。

4．do…while 循环

ActionScript 3.0 最可利用的循环语句是 do…while 语句。它与 while 语句十分相似，不过是当经过最初的循环之后，条件才被验证。其语法如下：

```
do
{
    //执行的语句块
}
while(表达式)
```

do…while 循环是 while 循环测试的扩展。do…while 语句与 while 语句唯一的区别在于，不管表达式的结果为真还是为假，循环语句至少执行一次。因此 do…while 循环适合于至少执

行一次循环体的情况。

5. for each…in 循环

for each…in 循环用于循环访问集合中的项目，它可以是 XML 或 XMLList 对象中的标签、对象属性保存的值或数组元素。例如，如代码 3.5 所示，可以使用 for each…in 循环来循环访问通用对象的属性，但是与 for…in 循环不同的是，for each…in 循环中的迭代变量包含属性所保存的值，而不包含属性的名称。

代码 3.5　使用 for each…in 循环输出对象的属性

```
var myObj:Object = {x:20, y:30};
for each (var num in myObj)
{
    trace(num);
}
```

上述代码的输出结果如下：

```
20
30
```

如果对象是密封类的实例，则将无法循环访问该对象的属性。即使是动态类的实例，也无法循环访问任何固定属性（即作为类定义的一部分定义的属性）。

for each…in 循环可以循环访问 XML 或 XMLList 对象，如代码 3.6 所示。

代码 3.6　使用 for each…in 循环输出 XML 中的元素

```
var myXML:XML = <users>
                    <fname>董红伟</fname>
                    <fname>李军华</fname>
                    <fname>李军民</fname>
                 </users>;
for each (var item in myXML.fname)
{
    trace(item);
}
```

上述代码的输出结果如下：

```
董红伟
李军华
李军民
```

for each…in 循环还可以循环访问数组中的元素，如代码 3.7 所示。

代码 3.7　使用 for each…in 循环输出数组中的元素

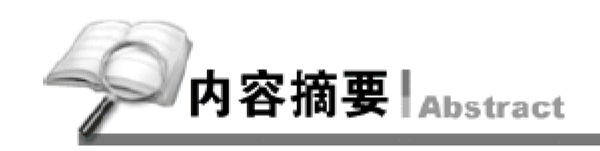
```
var myArray:Array = ["one", "two", "three"];
```

```
for each (var item in myArray)
{
    trace(item);
}
```


上述代码的输出结果如下：

```
one
two
hree
```

第4章 ActionScript 3.0 面向对象



内容摘要 Abstract

ActionScript 3.0 是一种面向对象的编程语言，即使是最基本的数据类型，如 Boolean、Number 和 String 类型，都属于类。面向对象编程是一种功能强大的程序设计方法，也可以说是一种编程思想。它关注应用程序的数据和处理数据所需要的方法。面向对象编程技术中使用了传统编程技术中所有的概念，例如变量、过程或函数调用，程序控制结构。面向对象在传统编程基础上还增加了新的概念，例如多态、继承、封装等。

面向对象程序设计以“数据控制访问代码”为主要原则，围绕数据来组织程序。在进行面向对象编程时需要定义数据和作用于数据上的方法。这样，数据类型可以精确地定义出哪种类型的操作可以应用于该数据。类、对象和方法是面向对象编程的基础。在类中定义了数据和实现这些数据的代码。本章首先阐述了面向对象的概念，然后介绍了在 ActionScript 3.0 中如何使用面向对象，包括对象、类，包、命名空间、枚举类，以及继承和接口等。

学习目标 Objective

- 理解面向对象思想
- 熟练地创建类和对象
- 掌握类成员
- 能够熟练地定义方法和调用方法
- 掌握如何使用包和命名空间
- 了解包和命名空间的异同
- 了解枚举类
- 掌握类的继承
- 了解接口并能够定义和实现接口

4.1 类和对象

面向对象程序开发是一种软件开发思想，它将数据和对数据的操作作为一个相互依赖、不可分割的整体。也就是说，把对数据的操作作为一个对象进行处理，采用数据抽象和信息隐蔽技术使软件开发简单化。这种思想更符合人们的思维习惯，有助于控制软件的复杂性，提高软件的生产效率，从而得到了广泛的应用，已经成为目前最为流行的一种软件开发方法

之一。在面向对象的内容中主要涉及到对象、类及传递的消息等，本节介绍类和对象。

4.1.1 面向对象概述

在面向对象的语言中，一切皆是对象，一个对象就代表一个具体的功能操作。也可以说面向对象编程是以术语对象来模拟现实世界，世界上的每件事物都可以模拟为对象，例如，一个人、一辆汽车等。我们不需要了解这个对象是如何实现某个操作的，只需要知道该对象可以完成哪个操作即可。编写出来的程序只不过是由对象堆砌而成的，这些对象可以是独立的，也可以是从另外一个对象继承过来的，对象之间可以传递消息，并通过消息来改变自身的状态。

面向对象思想的精要在于一切都是对象的意义。软件开发与设计是围绕着开发的目标进行的，即围绕这对象开发设计。一般而言，一个对象应具有属性和行为，以现实世界为例，“人”作为一种特殊的动物，也是我们所谓的“对象”。这个对象具有很多属性，例如，姓名、身高、体重、民族、国籍、出生年月、身份等。而行为呢，可以是行走、吃饭、跑步、乃至于玩游戏、踢足球。然而对象也有特殊的情况，或者只具有属性，或者只具有行为。例如身份证，可能只具有姓名、出生年月、籍贯、身份证号和性别等属性，而没有行为。

面向对象思想有三大机制，也可以说是类（把数据和函数包装在一个单独的单元，成为类）的三大机制，它们分别是封装、继承和多态。下面详细介绍面向对象思想的三大机制。

封装

一般来说封装就是隐藏类的数据成员，只向外提供一些公用的操作接口，只能通过这些接口来操作类的数据成员，而不能直接对这些数据成员进行赋值、改变等操作。这样做的好处在于如果选择直接暴露数据成员，有些人可能会把这些成员修改为非法数据导致程序出错。所以，封装操作数据成员的细节能够对新值进行验证或者其他操作。封装是将代码及其处理的数据绑定在一起的一种编程机制，该机制保证了程序和数据都不受外部干扰且不被误用。

以日常办公用的计算机为例，可以考虑计算机的硬盘。硬盘作为一个存储数据的容器，它是如何存储信息，以何种方式存储，对这些都不需要了解。只要知道是用来存储信息的，以及可以对硬盘进行哪些操作即可。当然，用户是不被允许去破坏硬盘本身的构造的。

与此相同的观点能被用于编程。封装代码的好处是每个人都知道怎么访问它，但却不必考虑它的内部实现细节，也不必害怕使用不当会带来负面影响。在 ActionScript 3.0 中，最基本的封装单元是类，一个类被定义为由一组对象所共享的行为，一个类的每个对象均包含它所定义的结构与行为。由于类的用途是封装复杂性，所以类的内部有隐藏实现复杂性的机制。ActionScript 3.0 中提供了私有和公有的访问模式，类的公有接口代表外部的用户应该知道或可以知道的实现细节，私有的方法、数据只能通过该类的成员方法来访问。这样可以确保不会发生关键数据被修改的事情。

继承

继承是面向对象程序设计的主要特征之一，它可以让开发人员重用代码，可以节省程序设计的时间。继承就是在类之间建立一种相交关系，使得新定义的派生类的实例可以继承已有父类的特征和能力，而且可以加入新的特性或者是修改已有的特性建立起类的新层次。继承也可以指一个对象从另一个对象中获得属性的过程，它支持按层次分类的概念。例如，公共汽车是汽车的一种，汽车又是车的一种，车又是机械的一种。如果不使用层次的概念，每

个对象都需要明确定义各自的全部特征；如果通过层次分类方式，一个对象只需要在它的类中定义它所特有的属性和行为，然后从超类中继承通用属性。因此，正是由于继承机制，才使得一个对象可以成为一个通用类的一个特定实例，一个深度继承的子类将继承它在类层次中的每个祖先的所有公有属性。

继承与封装可以互相作用从而提高了软件模块的可复用性和可扩充性，还可以提高软件的开发效率，也能够利用前人或自己以前的开发成果，同时在自己的开发过程中还能够有足够的灵活性，不拘泥于复用的模块。例如，一个给定的类封装了某些属性，它的任何子类将会含有同样的属性，另加各个子类所特有的属性。这是面向对象程序在复杂性上呈线性而非几何增长的一个重要概念，新的子类继承其所有祖先的所有公有属性，子类和系统中的其他代码不会产生无法预料的交互作用。

多态

同一操作作用于不同的对象，可以有不同的解释，产生不同的执行结果，这就是多态性。多态性通过派生类覆盖父类中的虚函数型方法来实现。多态性分为两种，一种是编译时的多态性，一种是运行时的多态性。下面分别介绍这两种多态性。

编译时的多态性 编译时的多态性是通过重载来实现的。对于非虚的成员来说，系统在编译时，根据传递的参数、返回的类型等信息决定实现何种操作。

运行时的多态性 运行时的多态性就是指直到系统运行时，才根据实际情况决定实现何种操作。ActionScript 3.0 中运行时的多态性是通过重写虚成员实现。

多态性的概念经常被说成是“一个接口，多种方法”。这意味着可以为一组相关的动作设计一个通用的接口。多态性允许同一个接口被同一类的多个动作使用，这样就降低了程序的复杂性。选择应用于每一种情形特定的动作（即类中的方法）是编译器的任务，程序员无须手工进行选择，只需记住并且使用通用接口即可。

多个类可以从单个父类“继承”。通过继承，类在父类所在的同一实现中接收父类的所有方法、属性和事件。这样，便可根据需要来实现附加成员，而且可以重写虚成员以提供不同的实现。这里要注意，继承类也可以实现接口，这两种技术不是互斥的。ActionScript 3.0 通过继承提供多态性。对于小规模开发任务而言，这是一个功能强大的机制，但对于大规模系统，通常会存在问题。过分强调继承驱动的多态性一般会导致资源大规模地从编码转移到设计，这对于缩短总的开发时间没有任何帮助。

对象是 ActionScript 3.0 语言的核心，是 ActionScript 3.0 语言的基本构造块。声明的每个变量、所编写的每个函数以及所创建的每个类实例都是一个对象。可以将 ActionScript 3.0 程序视为一组执行任务、响应事件以及相互通信的对象。

熟悉 Java 或 C++中面向对象编程（OOP）思想的读者，可能会将对象视为包含以下两类成员的模块：存储在成员变量或属性中的数据，以及可通过方法访问的行为。ECMAScript 第 4 版草案（ActionScript 3.0 所基于的标准）以相似但稍有不同的方式定义对象。在 ECMAScript 草案中，对象只是属性的集合。这些属性是一些容器，除了保存数据，还保存函数或其他对象。以这种方式附加到对象的函数称为方法。

在旧版本中，高级 ActionScript 程序员可以用特殊的内置语言元素来直接操作原型链。现在，由于 ActionScript 语言为基于类的编程接口提供了更成熟的实现，因此其中的许多特殊语言元素（如__proto__和__resolve）不再是该语言的一部分。而且，内部继承机制的优化还排

除了对继承机制的直接访问，从而大大改善了 Flash Player 的性能。

4.1.2　类的基本概念

通过上节的介绍，了解到对象是客观世界具体事物的实例，而类是客观世界的具有共同性质实体的抽象表示。也可以说类是对象的抽象和概括，是封装的基本单位。类是一种复杂的数据类型，它是将不同类型的数据和与这些数据相关的操作封装在一起的集合体。

具体来说，ActionScript 3.0 的类是一种对数据成员、函数成员和嵌套类型进行封装的数据结构。实体之间发生作用的实体的动作抽象为类的方法。一个方法作用于一个类的实体（也可以说是类的对象）后，实体的状态发生改变，永远都是说是对象的状态发生改变，而很少说类的状态发生改变。其中类的数据成员可以是常量、变量，函数成员可以是方法、属性、索引器、事件、操作符、实例构建器、静态构建器。除了某些导入的外部方法，类及其成员在 ActionScript 3.0 中的声明和实现通常要放在一起。

类是 ActionScript 3.0 中功能最为强大的数据类型。类定义了数据类型的数据和行为。然后，程序员可以创建作为此类的实例的对象。类支持继承，继承是面向对象编程的基础部分。创建类是使用 class 关键字来定义的，如下面的例子所示。

```
public class Person
{
    //此处定义类主体
}
```

早在 ActionScript 1.0 中，ActionScript 程序员就能使用 Function 对象创建类似类的构造函数。在 ActionScript 2.0 中，通过使用 class 和 extends 等关键字，添加了对类的支持。ActionScript 3.0 不但继续支持 ActionScript 2.0 中引入的关键字，而且还添加了一些新功能，如通过 protected 和 internal 修饰符增强了访问控制，通过 final 和 override 修饰符增强了对继承的控制。

如上代码中所示，关键字 class 前面的修饰符为访问级别。在该示例中，使用了 public，这表示任何人都可以基于该类创建对象。类的名称位于 class 关键字的后面，该示例的类名称为 Person。定义的其余部分是类的主体，用于定义行为和数据。类的字段、属性、方法和事件统称为“类成员”。

在该示例中使用了 public 访问修饰符，修饰符用于修改类型和类型成员的声明，在 ActionScript 3.0 中除了该修饰符外，还有其他类修饰符。表 4-1 中列出了类的修饰符。

表 4-1　类修饰符

修饰符	说明	修饰符	说明
dynamic	允许在运行时向实例添加属性	internal（默认）	对当前包内的引用可见
final	不得由其他类扩展	public	对所有位置的引用可见

使用 internal 以外的修饰符时，必须显式包含该修饰符才能获得相关的行为。例如，如果定义类时未包含 dynamic 修饰符，则不能在运行时向类实例中添加属性。列表中未包含名为 abstract 的修饰符，这是因为 ActionScript 3.0 不支持抽象类。同时，列表中未包含名为 private 和 protected 的修饰符。这两个修饰符只在类成员定义中使用，不可以应用于类本身。

4.1.3 类成员修饰符

类的成员包括变量、常量和方法，在 ActionScript 3.0 中，提供了可以与类的任何成员一起使用的修饰符，这些修饰符分为访问控制修饰符和 static 修饰符。表 4-2 中列出了类成员修饰符及其说明。

表 4-2 类成员修饰符

修饰符	说明
internal（默认）	对同一包中的引用可见
private	对同一类中的引用可见
protected	对同一类及派生类中的引用可见
public	对所有位置的引用可见
static	指定某一成员属于该类，而不属于该类的实例
override	覆盖超类的同名方法
final	禁止子类覆盖该方法

ActionScript 3.0 提供了 4 个特殊的修饰符，来控制对类中定义成员的访问，public、private、protected 和 internal。

使用 public 修饰符可使某一成员在脚本的任何位置可见。例如，要使某个方法可用于包外部的代码，必须使用 public 修饰符声明该方法。这适用于任何成员，不管成员是使用 var、const 还是 function 关键字声明的。

使用 private 修饰符，可使某一成员，只对成员的定义类中的调用方可见。这一行为不同于 ActionScript 2.0 中 private 修饰符的行为，后者允许子类访问超类中的私有成员。另一处明显的行为变化是，能否在运行时访问。在 ActionScript 2.0 中，private 关键字只在编译时禁止访问，运行时很容易避开它。在 ActionScript 3.0 中，这种情况不复存在，标记为 private 的成员在编译时和运行时都不可用。

例如，代码 4.1 中创建了名为 PrivateExample 的简单类，其中包含一个私有变量，然后尝试从该类的外部访问该私有变量。在 ActionScript 2.0 中，编译时访问被禁止，但是使用属性访问运算符（[]）可以很容易地避开，属性访问运算符在运行时（而不是编译时）执行属性查找。

代码 4.1 使用属性访问运算符访问私有变量

```
class PrivateExample
{
    private var privVar:String = "private variable";
}
var myExample:PrivateExample = new PrivateExample();
// 在严格模式下发生编译时错误
trace(myExample.privVar);
// ActionScript 2.0 允许访问，但在 ActionScript 3.0 中，这是一个运行时错误
trace(myExample["privVar"]);
```

在 ActionScript 3.0 中使用严格模式时，尝试使用点运算符（myExample.privVar）访问私有成员会导致编译时错误。否则，会在运行时报告错误，就像使用属性访问运算符（myExample["privVar"]）一样。

在使用 dynamic 修饰符声明的类中，尝试访问私有变量时，不会导致运行时错误，只是变量不可见，Flash Player 返回值为 undefined。但是，如果在严格模式下使用点运算符访问私有成员，则会发生编译时错误。代码 4.2 与代码 4.1 相同，只是 PrivateExample 类被声明为动态类。

代码 4.2　修改后的 PrivateExample 类

```
dynamic class PrivateExample
{
    private var privVar:String = "private variable";
}
var myExample:PrivateExample = new PrivateExample();
// 在严格模式下发生编译时错误
trace(myExample.privVar);
// 输出: undefined
trace(myExample["privVar"]);
```

当类外部的代码尝试访问 private 成员时，动态类通常会返回值 undefined，而不是生成错误。表 4-3 说明了只有在严格模式下，使用点运算符访问 private 成员时才会生成错误。

表 4-3　类外部的代码访问 private 成员时，不同运算符返回的值

运算符	严格模式	标准模式
点运算符（.）	编译时错误	undefined
中括号运算符（[]）	undefined	undefined

protected 修饰符是 ActionScript 3.0 的新增修饰符，可使成员对所属类或子类中的调用方可见。换句话说，protected 成员在所属类中可用，或者在继承层次结构中，该类下面的类中可用。无论子类在同一包中还是在不同包中，这一点都适用。

对于熟悉 ActionScript 2.0 的用户而言，此功能类似于 ActionScript 2.0 中的 private 修饰符。ActionScript 3.0 中的 protected 修饰符还类似于 Java 中的 protected 修饰符，不同之处在于，在 Java 中，该 protected 成员还允许访问同一包中的调用方。如果存在子类需要的变量或方法，但要对继承链外部的代码隐藏该变量或方法，此时 protected 修饰符会很有用。

internal 修饰符是 ActionScript 3.0 的新增修饰符，可使成员对所在包中的调用方可见。该修饰符是包中代码的默认修饰符。

internal 修饰符与 Java 中的默认访问控制相似，不过，在 Java 中该访问级别没有明确的名称，只能通过省略所有其他访问修饰符来实现。ActionScript 3.0 中提供的 internal 修饰符，旨在为程序员提供一个明确表达自己意图的选项，以使成员仅对所在包中的调用方可见。

static 修饰符可以与用 var、const 或 function 关键字声明的那些成员一起使用，使用该修饰符可将成员附加到类而不是类的实例。类外部的代码必须使用类名（而不是使用实例名）调用静态成员。

静态属性不由子类继承，但是这些属性是子类作用域链中的一部分。这意味着在子类体中，不必引用在其中定义静态变量或方法的类，就可以使用静态变量或方法。

4.1.4 定义方法

方法是类定义中的函数。创建类的一个实例后，该实例就会捆绑一个方法。与在类外部声明的函数不同，不能将方法与附加方法的实例分开使用。

方法是使用 function 关键字定义的。可以使用函数语句定义，如下所示。

```
public function sampleFunction():String {}
```

或者，也可以使用分配了函数表达式的变量，如下所示。

```
public var sampleFunction:Function = function () {}
```

多数情况下，需要使用函数语句而不是函数表达式，原因如下。

① 函数语句更为简洁易读。

② 函数语句允许使用 override 和 final 关键字。有关详细信息，请参阅覆盖方法。

③ 函数语句在标识符（即函数名）与方法体代码之间创建了更强的绑定。由于可以使用赋值语句更改变量值，可随时断开变量与其函数表达式之间的连接。虽然可通过使用 const（不是 var）声明变量来解决这个问题，但这种方法并不是最好的做法，因为这会使代码难以阅读，还会禁止使用 override 和 final 关键字。

1. 构造函数

构造函数是与在其中定义函数的类共用同一名称的函数。只要使用 new 关键字创建了类实例，就会执行构造函数方法中包括的所有代码。例如，代码 4.3 中定义名为 Example 的简单类，该类包含名为 status 的属性。status 变量的初始值是在构造函数中设置的。

代码 4.3 在构造函数设置属性

```
class Example
{
    public var status:String;
    public function Example()
    {
        status = "已初始化";
    }
}
var myExample:Example = new Example();
trace(myExample.status);
```

代码 4.3 的输出结果如下所示。

```
已初始化
```

构造函数只能是公共方法，也就是只能使用 public 修饰符。不能对构造函数使用任何其他访问控制修饰符（包括 private、protected 或 internal）。

构造函数，可以使用 super()语句显式地调用其直接超类的构造函数。如果未显式调用超类构造函数，编译器会在构造函数体中的第一条语句前自动插入一个调用。还可以使用 super 前缀作为对超类的引用，来调用超类的方法。如果决定在同一构造函数中使用 super()和 super，务必先调用 super()。否则，super 引用的行为将会与预期不符。另外，super()构造函数也应在 throw 或 return 语句之前调用。

代码 4.4 说明如果在调用 super()构造函数之前尝试使用 super 引用，将会产生错误。新类 ExampleEx 扩展了 Example 类。ExampleEx()构造函数，尝试访问在其超类中定义的状态变量，但访问是在调用 super()之前进行的。ExampleEx()构造函数中的 trace()语句生成了 null 值，原因是 status 变量在 super()构造函数执行之前不可用。

代码 4.4　在调用 super()构造函数之前使用 super 引用

```
class ExampleEx extends Example
{
    public function ExampleEx()
    {
        trace(super.status);
        super();
    }
}
var mySample:ExampleEx = new ExampleEx();
```

虽然在构造函数中使用 return 语句是合法的，但是不允许返回值。换句话说，return 语句不得有相关的表达式或值。因此，不允许构造函数方法返回值，这意味着不可以指定任何返回值。

如果没有在类中定义构造函数方法，编译器将会自动创建一个空构造函数。如果某个类扩展了另一个类，编译器将会在所生成的构造函数中包括 super()调用。ActionScript 3.0 不支持对构造函数的重载。

2．静态方法

静态方法也叫作类方法，是使用 static 修饰符声明的方法。静态方法附加到类而不是类的实例，因此在封装对单个实例的状态以外的内容有影响的功能时，静态方法很有用。由于静态方法附加到整个类，所以只能通过类访问静态方法，而不能通过类实例访问。

静态方法为封装所提供的功能，不仅仅在影响类实例状态的方面。换句话说，如果方法提供的功能对类实例的值没有直接的影响，该方法应是静态方法。例如，Date 类具有名为 parse()的静态方法，它接收字符串并将其转换为数字。该方法就是静态方法，因为它并不影响类的单个实例。而 parse()方法使用表示日期值的字符串分析该字符串，然后使用与 Date 对象的内部表示形式兼容的格式返回一个数字。此方法不是实例方法，因为将该方法应用到 Date 类的实例并没有任何意义。

将静态 parse()方法与 Date 类的一个实例方法（如 getMonth()）相比较。getMonth()方法是一个实例方法，因为它通过检索 Date 实例的特定对象，即 month，对实例值直接执行操作。

由于静态方法不绑定到单个实例，因此不能在静态方法体中使用关键字 this 或 super。this 和 super 这两个引用只在实例方法上下文中有意义。



与其他基于类的编程语言不同，ActionScript 3.0 中的静态方法不可以继承。

3. 实例方法

实例方法指的是不使用 static 修饰符声明的方法。实例方法附加到类实例而不是整个类，在实现对类的各个实例有影响的功能时，实例方法很有用。例如，Array 类包含名为 sort()的实例方法，该实例方法直接对 Array 实例执行操作。

在实例方法体中，静态变量和实例变量都在作用域中，这表示使用一个简单的标识符可以引用同一类中定义的变量。例如，代码 4.5 中，类 CustomArray 扩展了 Array 类。CustomArray 类定义一个名为 arrayCountTotal 的静态变量（用于跟踪类实例总数）、一个名为 arrayNumber 实例变量（用于跟踪创建实例的顺序）和一个名为 getPosition()的实例方法（用于返回这两个变量的值）。

代码 4.5　实例方法中的变量

```
public class CustomArray extends Array
{
    public static var arrayCountTotal:int = 0;
    public var arrayNumber:int;
    public function CustomArray()
    {
        arrayNumber = ++arrayCountTotal;
    }
    public function getPosition():String
    {
         return ("Array " + arrayNumber + " of " + arrayCountTotal);
    }
}
```

虽然类外部的代码必须使用 CustomArray.arrayCountTotal 通过类对象来引用 arrayCountTotal 静态变量，但是位于 getPosition()方法体中的代码可以直接引用。即使对于超类中的静态变量，这一点也适用。虽然在 ActionScript 3.0 中不继承静态成员，但是超类的静态成员在作用域中。例如，Array 类有几个静态变量，其中一个是名为 DESCENDING 的常量。位于 Array 子类中的代码可以使用一个简单的标识符，来引用静态常量 DESCENDING，如代码 4.6 所示。

代码 4.6　实例方法中引用超类的静态成员

```
public class CustomArray extends Array
```

```
{
    public function testStatic():void
    {
        trace(DESCENDING);
    }
}
```


代码 4.6 的输出结果如下所示。

```
2
```

实例方法体中的 this 引用的值是对方法所附加实例的引用。代码 4.7 说明 this 引用指向包含方法的实例。

代码 4.7　this 引用指向包含方法的实例

```
class ThisTest
{
    function thisValue():ThisTest
    {
        return this;
    }
}
var myTest:ThisTest = new ThisTest();
trace(myTest.thisValue() == myTest);
```

代码 4.7 的输出结果如下所示。

```
true
```

使用关键字 override 和 final 可以控制实例方法的继承。可以使用 override 修饰符重新定义继承的方法，以及使用 final 修饰符禁止子类覆盖方法。

4. get 和 set 存取器方法

get 和 set 存取器方法也称为 getter 和 setter，可以使用这些函数为创建的类提供易于使用的编程接口，并遵循信息隐藏和封装的编程原则。使用 get 和 set 函数可保持类的私有属性，但允许类用户访问这些属性，就像它们在访问类变量而不是调用类方法一样。

这种方法的好处是，可避免出现具有不实用名称的传统存取器方法，如 getPropertyName()和 setPropertyName()。getter 和 setter 的另一个好处是，可避免允许进行读写访问的每个属性，有两个面向公共的函数。

代码 4.8 中，类名为 GetSet，其中包含名为 publicAccess()的 get 和 set 存取器方法，用于提供对名为 privateProperty 的私有变量的访问。

代码 4.8　创建 get 和 set 存取器方法

```
class GetSet
{
```

```
    private var privateProperty:String;
    public function get publicAccess():String
    {
        return privateProperty;
    }
    public function set publicAccess(setValue:String):void
    {
        privateProperty = setValue;
    }
}
```

如果尝试直接访问属性 privateProperty，将会发生错误，如下所示。

```
var myGetSet:GetSet = new GetSet();
// 发生错误
trace(myGetSet.privateProperty);
```

程序员使用 GetSet 类的 publicAccess 方法，实际上是对名为 privateProperty 的 private 属性执行的一对 get 和 set 存取器方法。代码 4.9 将实例化 GetSet 类，然后使用名为 publicAccess 的公共存取器，设置 privateProperty 的值。

代码 4.9　调用存取器方法

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.publicAccess);
myGetSet.publicAccess = "hello";
trace(myGetSet.publicAccess);
```

代码 4.9 的输出结果如下所示：

```
null
hello
```

使用存取器方法，还可以覆盖从超类继承来的属性，这是使用常规类成员变量时不能做到的。在子类中不能覆盖使用 var 关键字声明的类成员变量。但是，使用存取器方法创建的属性没有此限制。可以对从超类继承的存取器方法使用 override 修饰符。

5. 绑定方法

绑定方法有时也叫作闭包方法，这种方法作为参数传递给函数，或作为值从函数返回。在 ActionScript 3.0 中，新增的绑定方法类似于闭包函数，其中保留了词汇环境，即使从其实例中提取出来也是如此。绑定方法与闭包函数之间的主要不同差别是，绑定函数的 this 引用，保留到实现方法的实例的链接或绑定。换句话说，绑定方法中的 this 引用，总是指向实现方法的原始对象。对于闭包函数，this 引用是通用的，这意味着调用函数时，该引用指向与函数关联的任何对象。

如果使用 this 关键字，了解绑定方法就很重要。重新调用 this 关键字可提供对方法父对象的引用。大多数 ActionScript 程序员都希望 this 关键字总是引用包含方法定义的对象或类。

但是，如果不使用绑定方法，并不总是做到这样。例如，在以前版本的 ActionScript 中，this 引用并不总是引用实现方法的实例。从 ActionScript 2.0 的实例中提取方法后，不但 this 引用不绑定到原始实例，而且实例类的成员变量和方法也不可用。在 ActionScript 3.0 中不存在这样的问题，这是因为将方法当作参数传递时会自动创建绑定方法。绑定方法用于确保 this 关键字总是引用在其中定义了方法的对象或类。

代码 4.10 定义了名为 ThisTest 的类，该类包含一个名为 foo()的方法（该方法定义为绑定方法）和一个名为 bar()的方法（该方法返回绑定方法）。类外部的代码创建 ThisTest 类的实例，然后调用 bar()方法，最后将返回值存储在名为 myFunc 的变量中。

代码 4.10　定义和返回绑定方法

```
class ThisTest
{
   private var num:Number = 3;
   // 定义绑定方法
   function foo():void
   {
      trace("foo's this: " + this);
      trace("num: " + num);
   }
   function bar():Function
   {
      // 返回绑定方法
      return foo;
   }
}

var myTest:ThisTest = new ThisTest();
var myFunc:Function = myTest.bar();
trace(this);
myFunc();
```

代码 4.10 的输出结果如下所示。

```
foo's this: [object ThisTest]
output: num: 3
```

代码 4.10 的最后两行表明：虽然前一行中的 this 引用指向全局对象，但绑定方法 foo()中的 this 引用仍然指向 ThisTest 类的实例。另外，存储在 myFunc 变量中的绑定方法仍然可以访问 ThisTest 类的成员变量。如果代码 4.10 在 ActionScript 2.0 中运行，this 引用会匹配，但 num 变量将为 undefined。

绑定方法最值得注意的一种情况是使用事件处理函数，因为 addEventListener()方法要求将函数或方法作为参数来传递。

4.1.5 定义属性

属性是提供对对象或类的特性进行访问的成员。可以使用 var 或 const 关键字声明属性。在脚本的整个执行过程中，使用 var 关键字声明的属性可多次更改其值。使用 const 关键字声明的属性值称为常量，只能赋值一次。尝试给已初始化的常量分配新值，将生成错误。

1. 静态变量

静态变量是使用 static 修饰符和 var 或 const 语句共同声明的。静态变量附加到类而不是类的实例，对于存储和共享应用于对象的整个类的信息非常有用。例如，当要保存类实例化的总次数或者要存储允许的最大类实例数，使用静态变量比较合适。

下面的示例创建一个 totalCount 变量（用于跟踪类实例化次数）和一个 MAX_NUM 常量（用于存储最大实例化数）。totalCount 和 MAX_NUM 这两个属性是静态变量，因为它们包含的值应用于整个类，而不是某个特定实例。

```
class StaticVars
{
    public static var totalCount:int = 0;
    public static const MAX_NUM:uint = 16;
}
```

StaticVars 类及其任何子类外部的代码只能通过该类本身来引用 totalCount 和 MAX_NUM 属性。例如，以下代码输出的结果为 0 和 16。

```
trace(StaticVars.totalCount);   // 输出: 0
trace(StaticVars.MAX_NUM);      // 输出: 16
```

不能通过类实例访问静态变量，以下代码会返回错误。

```
var myStaticVars:StaticVars = new StaticVars();
trace(myStaticVars.totalCount); // 错误
trace(myStaticVars.MAX_NUM);    // 错误
```

必须在声明常量的同时，初始化使用 static 和 const 修饰符声明的变量，就像 StaticVars 类初始化 MAX_NUM 那样。不能为构造函数或实例方法中的 MAX_NUM 赋值。以下代码会生成错误，因为它不是初始化静态常量的有效方法。

```
// !! 采用这种方法初始化静态常量将发生错误
class StaticVars2
{
    public static const UNIQUESORT:uint;
    function initializeStatic():void
    {
        UNIQUESORT = 16;
    }
}
```

2．实例变量

实例变量包括使用 var 和 const 语句，但未使用 static 修饰符声明的属性。实例变量附加到类实例而不是整个类，对于存储特定于实例的值很有用。例如，Array 类有一个名为 length 的实例属性，用来存储 Array 类的特定实例保存的数组元素的个数。

不能覆盖子类中声明为 var 或 const 的实例变量。但是，通过覆盖 get 和 set 存取器方法，可以实现类似于覆盖变量的功能。

4.2　包和命名空间

包和命名空间是两个相关的概念。使用包，可以以有利于共享代码，并尽可能减少命名冲突的方式将多个类定义捆绑在一起。使用命名空间可以控制标识符（如属性名和方法名）的可见性。无论命名空间位于包的内部还是外部，都可以应用于代码。包可用于组织类文件，命名空间可用于管理各个属性和方法的可见性。

4.2.1　包

在 ActionScript 3.0 中，包是用命名空间实现的，但包和命名空间并不同义。在声明包时，可以隐式创建一个特殊类型的命名空间并保证它在编译时是已知的。显式创建的命名空间在编译时不必是已知的。

代码 4.11 中，使用 package 指令，来创建一个包含单个类的简单包。

代码 4.11　使用 package 指令创建包

```
package samples
{
    public class SampleCode
    {
        public var sampleGreeting:String;
        public function sampleFunction()
        {
            trace(sampleGreeting + " from sampleFunction()");
        }
    }
}
```

在代码 4.1 中，该类的名称是 SampleCode。由于该类位于 samples 包中，因此编译器在编译时，会自动将其类名称限定为完全限定名称：samples.SampleCode。编译器还限定任何属性或方法的名称，以使 sampleGreeting 和 sampleFunction() 分别变成 samples.SampleCode.sampleGreeting 和 samples.SampleCode.sampleFunction()。

许多开发人员（尤其是那些具有 Java 编程背景的人）可能会选择只将类放在包的顶级。

但是，ActionScript 3.0 不但支持将类放在包的顶级，而且还支持将变量、函数甚至语句放在包的顶级。此功能的一个高级用法是，在包的顶级定义一个命名空间，以便它对于该包中的所有类均可用。但是，请注意，在包的顶级只允许使用两个访问说明符：public 和 internal。Java 允许将嵌套类声明为私有，而 ActionScript 3.0 则不同，它既不支持嵌套类也不支持私有类。

但是，在其他许多方面，ActionScript 3.0 中的包与 Java 编程语言中的包非常相似。从代码 4.11 可以看出，完全限定的包，应用点运算符（.）来表示，这与 Java 相同。可以用包将代码组织成直观的分层结构，以供其他程序员使用。这样，就可以将自己所创建的包与他人共享，还可以在自己的代码中使用他人创建的包，从而推动了代码共享。

使用包还有助于确保所使用的标识符名称是唯一的，而且不与其他标识符名称冲突。事实上，有些人认为这才是包的主要优点。例如，假设两个希望相互共享代码的程序员各创建了一个名为 SampleCode 的类。如果没有包，就会造成名称冲突，唯一的解决方法就是重命名其中的一个类。但是，使用包就可以将其中的一个（最好是两个）类放在具有唯一名称的包中，从而轻松地避免了名称冲突。

还可以在包名称中嵌入点来创建嵌套包，这样就可以创建包的分层结构。Flash Player API 提供的 flash.xml 包就是一个很好的例子。flash.xml 包嵌套在 Flash 包中。

flash.xml 包中包含在早期的 ActionScript 版本中使用的旧 XML 分析器。该分析器现在之所以包含在 flash.xml 包中，原因之一是，旧 XML 类的名称与一个新 XML 类的名称冲突，这个新 XML 类实现 ActionScript 3.0 中的 XML for ECMAScript (E4X)规范功能。

尽管首先将旧的 XML 类移入包中是一个不错的主意，但是使用旧 XML 类的用户，都必须导入 flash.xml 包。这样，使用旧 XML 类，除非总是记得其完全限定名称（flash.xml.XML），否则同样会造成名称冲突。为避免这种情况，现在已将旧 XML 类，命名为 XMLDocument，如下面的代码所示。

```
package flash.xml
{
    class XMLDocument {}
    class XMLNode {}
    class XMLSocket {}
}
```

大多数 Flash Player API 都划分到 flash 包中。例如，flash.display 包中包含显示列表 API，flash.events 包中包含新的事件模型。

1. 创建包

ActionScript 3.0 在包、类和源文件的组织方式上具有很大的灵活性。早期的 ActionScript 版本只允许每个源文件有一个类，而且要求源文件的名称与类名称匹配。ActionScript 3.0 允许在一个源文件中包括多个类，但是，每个文件中只有一个类可供该文件外部的代码使用。换言之，每个文件中只有一个类可以在包中进行声明。必须在包定义的外部声明其他任何类，以使这些类对于该源文件外部的代码不可见。在包中声明类的名称必须与源文件的名称匹配。

ActionScript 3.0 在包的声明方式上也具有更大的灵活性。在早期的 ActionScript 版本中，包只是表示可用来存放源文件的目录，不必用 package 语句来声明包，而是在类声明中将包名

称包括在完全限定的类名称中。在 ActionScript 3.0 中，尽管包仍表示目录，但是它现在不只包含类。在 ActionScript 3.0 中，使用 package 语句来声明包，这意味着还可以在包的顶级声明变量、函数和命名空间，甚至还可以在包的顶级包括可执行语句。如果在包的顶级声明变量、函数或命名空间，则在顶级只能使用 public 和 internal 修饰符，并且每个文件中只能有一个包级声明使用 public 修饰符（无论该声明是类声明、变量声明、函数声明还是命名空间声明）。

包的作用是组织代码并防止名称冲突。不应将包的概念与类继承这一不相关的概念混淆。位于同一个包中的两个类具有共同的命名空间，但是它们在其他任何方面都不必相关。

2. 导入包

如果希望使用位于某个包内部的特定类，则必须导入该包或该类。这与 ActionScript 2.0 不同，在 ActionScript 2.0 中，类的导入是可选的。

以代码 4.11 中的 SampleCode 类为例。如果该类位于名为 samples 的包中，那么，在使用 SampleCode 类之前，必须使用下列导入语句之一：

```
import samples.*;
```

或者只导入 SampleCode 类：

```
import samples.SampleCode;
```

通常，import 语句越具体越好。如果只打算使用 samples 包中的 SampleCode 类，则应只导入 SampleCode 类，而不应导入该类所属的整个包。导入整个包可能会导致意外的名称冲突。

还必须将定义包或类的源代码放在类路径内部。类路径是用户定义的本地目录路径列表，它决定了编译器将在何处搜索导入的包和类。类路径有时称为生成路径或源路径。

在正确地导入类或包之后，可以使用类的完全限定名称（samples.SampleCode），也可以只使用类名称本身（SampleCode）。

当同名的类、方法或属性会导致代码不明确时，完全限定的名称是非常有用的。但是，如果将它用于所有的标识符，则会使代码变得难以管理。例如，在实例化 SampleCode 类的实例时，使用完全限定的名称会导致代码冗长，如下代码所示。

```
var mySample:samples.SampleCode = new samples.SampleCode();
```

包的嵌套级别越高，代码的可读性越差。如果确信不明确的标识符不会导致问题，就可以通过使用简单的标识符，来提高代码的可读性。例如，如果在实例化 SampleCode 类的新实例时仅使用类标识符，代码就会简短得多，如下代码所示。

```
var mySample:SampleCode = new SampleCode();
```

如果尝试使用标识符名称，而不先导入相应的包或类，编译器将找不到类定义。另一方面，即便导入了包或类，只要尝试定义的名称与所导入的名称冲突，就会产生错误。

创建包时，该包的所有成员的默认访问说明符是 internal。这意味着，默认情况下，包成员仅对其所在包的其他成员可见。如果希望某个类对包外部的代码可用，则必须将该类声明为 public。例如，代码 4.12 和代码 4.13 中，samples 包中有 SampleCode 和 CodeFormatter 两

个类。

代码 4.12　SampleCode 类：SampleCode.as

```
// SampleCode.as 文件
package samples
{
    public class SampleCode {}
}
```

代码 4.13　CodeFormatter 类：CodeFormatter.as

```
// CodeFormatter.as 文件
package samples
{
    class CodeFormatter {}
}
```

SampleCode 类在包的外部可见，因为它被声明为 public 类。但是，CodeFormatter 类仅在 samples 包的内部可见。如果尝试在 samples 包外部访问 CodeFormatter 类，将会产生一个错误，如代码 4.14 所示。

代码 4.14　在 samples 包外部访问包中定义的类

```
import samples.SampleCode;
import samples.CodeFormatter;
// 正确，public 类
var mySample:SampleCode = new SampleCode();
// 错误
var myFormatter:CodeFormatter = new CodeFormatter();
```

如果希望这两个类在包外部均可用，必须将它们都声明为 public。不能将 public 修饰符应用于包声明。

完全限定的名称，可用来解决在使用包时可能发生的名称冲突。如果导入两个包，但它们用同一个标识符来定义类，就可能会发生名称冲突。例如，考虑代码 4.15 中的包，该包也有一个名为 SampleCode 的类。

代码 4.15　在另外包中定义 SampleCode 类

```
package langref.samples
{
    public class SampleCode {}
}
```

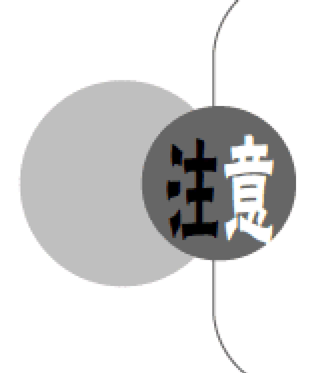
如果按如下方式导入两个类，在引用 SampleCode 类时将会发生名称冲突，代码如下所示。

```
import samples.SampleCode;
import langref.samples.SampleCode;
```

```
// 名称冲突
var mySample:SampleCode = new SampleCode();
```

编译器无法确定要使用哪个 SampleCode 类。要解决此冲突，必须使用每个类的完全限定名称，正确代码如下所示。


```
var sample1:samples.SampleCode = new samples.SampleCode();
var sample2:langref.samples.SampleCode = new langref.samples.SampleCode();
```



具有 C++背景的程序员通常会将 import 语句与#include 混淆。#include 指令在 C++中是必需的，因为 C++编译器一次处理一个文件，而且除非显式包括了头文件，否则将不会在其他文件中查找类定义。ActionScript 3.0 有一个 include 指令，但是它的作用不是为了导入类和包。要在 ActionScript 3.0 中导入类或包，必须使用 import 语句，并将包含该包的源文件放在类路径中。

4.2.2　命名空间

通过命名空间可以控制所创建的属性和方法的可见性。将 public、private、protected 和 internal 访问控制修饰符视为内置的命名空间。如果这些预定义的访问控制说明符无法满足要求，可以创建自己的命名空间。

如果熟悉 XML 命名空间，那么对本节讨论的大部分内容不会感到陌生，但是 ActionScript 实现的语法和细节与 XML 的稍有不同。即使以前从未使用过命名空间，也没有关系，因为命名空间概念本身很简单，但是其实现涉及一些需要了解的特定术语。

要了解命名空间的工作方式，有必要先了解属性或方法的名称总是包含两部分：标识符和命名空间。标识符通常被视为名称。例如，以下类定义中的标识符是 sampleGreeting 和 sampleFunction()。

```
class SampleCode
{
    var sampleGreeting:String;
    function sampleFunction () {
        trace(sampleGreeting + " from sampleFunction()");
    }
}
```

只要定义不以命名空间修饰符开头，就会用默认 internal 命名空间限定其名称，这意味着，它们仅对同一个包中的调用方可见。如果编译器设置为严格模式，则编译器会发出一个警告，指明 internal 命名空间将应用于没有命名空间属性的任何标识符。为了确保标识符可在任何位置使用，必须在标识符名称的前面明确加上 public 修饰符。在上面的示例代码中，sampleGreeting 和 sampleFunction()都有一个命名空间值 internal。

使用命名空间时，应遵循以下 3 个基本步骤。

第一，必须使用 namespace 关键字来定义命名空间。例如，下面的代码定义 version1 命名

空间。

```
namespace version1;
```

第二，在属性或方法声明中，使用命名空间（而非访问控制修饰符）来应用命名空间。下面的代码将一个名为 myFunction()的函数放在 version1 命名空间中。

```
version1 function myFunction() {}
```

第三，在应用了该命名空间后，可以使用 use 指令引用它，也可以使用该命名空间来限定标识符的名称。下面的代码通过 use 指令来引用 myFunction()函数。

```
use namespace version1;
myFunction();
```

还可以使用限定名称来引用 myFunction()函数，如下面的代码所示。

```
version1::myFunction();
```

1. 定义命名空间

命名空间中包含一个名为统一资源标识符（URI）的值，该值有时称为命名空间名称。使用 URI 可确保命名空间定义的唯一性。

可通过使用以下两种方法之一来声明命名空间，以创建命名空间：像定义 XML 命名空间那样使用显式 URI 定义命名空间；省略 URI。下面的代码说明如何使用 URI 来定义命名空间。

```
namespace flash_proxy = "http://www.adobe.com/flash/proxy";
```

URI 用作该命名空间的唯一标识字符串。如果省略 URI（如下面的代码所示），则编译器将创建一个唯一的内部标识字符串来代替 URI。对于这个内部标识字符串，不具有访问权限。

```
namespace flash_proxy;
```

在定义了命名空间（具有 URI 或没有 URI）后，就不能在同一个作用域内重新定义该命名空间。如果尝试定义的命名空间已在同一个作用域内定义过，则将生成编译器错误。

如果在某个包或类中定义了一个命名空间，则该命名空间可能对于此包或类外部的代码不可见，除非使用了相应的访问控制说明符。例如，下面的代码显示了在 flash.utils 包中定义的 flash_proxy 命名空间。在下面的代码中，缺乏访问控制说明符意味着 flash_proxy 命名空间将仅对于 flash.utils 包内部的代码可见，而对于该包外部的任何代码都不可见。

```
package flash.utils
{
    namespace flash_proxy;
}
```

下面的代码使用 public 属性以使 flash_proxy 命名空间对该包外部的代码可见。

```
package flash.utils
{
```

```
    public namespace flash_proxy;
}
```

2. 应用命名空间

应用命名空间意味着在命名空间中放置定义。可以放在命名空间中的定义包括函数、变量和常量（不能将类放在自定义命名空间中）。

例如，考虑一个使用 public 访问控制命名空间声明的函数。在函数的定义中使用 public 修饰符会将该函数放在 public 命名空间中，从而使该函数对于所有的代码都可用。在定义了某个命名空间之后，可以按照与使用 public 修饰符相同的方式来使用所定义的命名空间。该定义将对于可以引用自定义命名空间的代码可用。例如，如果定义一个名为 example1 的命名空间，则可以添加一个名为 myFunction()的方法并将 example1 用作修饰符，如下面的示例代码所示。

```
namespace example1;
class someClass
{
    example1 myFunction() {}
}
```

如果在声明 myFunction()方法时将 example1 命名空间用作修饰符，则意味着该方法属于 example1 命名空间。

在应用命名空间时，应切记以下几点。

对于每个声明只能应用一个命名空间。

不能一次将同一个命名空间修饰符应用于多个定义。换言之，如果希望将自己的命名空间应用于 10 个不同的函数，则必须将该命名空间作为修饰符分别添加到这 10 个函数的定义中。

如果应用了命名空间，则不能同时指定访问控制修饰符，因为命名空间和访问控制修饰符是互斥的。换言之，如果应用了命名空间，就不能将函数或属性声明为 public、private、protected 或 internal。

3. 引用命名空间

在使用借助于任何访问控制命名空间（如 public、private、protected 和 internal）声明的方法或属性时，无需显式引用命名空间。这是因为，对于这些特殊命名空间的访问，由上下文控制。例如，放在 private 命名空间中的定义，会自动对于同一个类中的代码可用。但是，对于自定义的命名空间，并不存在这样的上下文相关性。要使用已经放在某个自定义命名空间中的方法或属性，必须引用该命名空间。

可以用 use namespace 指令来引用命名空间，也可以使用名称限定符（::）来以命名空间限定名称。用 use namespace 指令引用命名空间会打开该命名空间，这样它便可以应用于任何未限定的标识符。例如，如果已经定义了 example1 命名空间，则可以通过使用 use namespace example1 来访问该命名空间中的名称，代码如下所示。

```
use namespace example1;
myFunction();
```

一次可以打开多个命名空间。在使用 use namespace 打开了某个命名空间之后，它会在打开它的整个代码块中保持打开状态。不能显式关闭命名空间。

但是，如果同时打开多个命名空间，则会增加发生名称冲突的可能性。如果不愿意打开命名空间，则可以用命名空间和名称限定符来限定方法或属性名，从而避免使用 use namespace 指令。例如，下面的代码说明如何用 example1 命名空间来限定 myFunction()名称。

```
example1::myFunction();
```

4. 使用命名空间

在 Flash Player API 中的 flash.utils.Proxy 类中，可以找到用来防止名称冲突的命名空间的实例。Proxy 类取代了 ActionScript 2.0 中的 Object.__resolve 属性，可用来截获对未定义的属性或方法的引用，以免发生错误。为了避免名称冲突，将 Proxy 类的所有方法都放在 flash_proxy 命名空间中。

为了更好地了解 flash_proxy 命名空间的使用方法，需要了解如何使用 Proxy 类。Proxy 类的功能仅对于继承它的类可用。换言之，如果要对某个对象使用 Proxy 类的方法，则该对象的类定义必须是对 Proxy 类的扩展。例如，如果希望截获对未定义的方法的调用，则应扩展 Proxy 类，然后覆盖 Proxy 类的 callProperty()方法。

前面已讲到，实现命名空间的过程通常分为 3 步，即定义、应用然后引用命名空间。但是，由于从不显式调用 Proxy 类的任何方法，因此只是定义和应用 flash_proxy 命名空间，而不用引用它。Flash Player API 定义 flash_proxy 命名空间，并在 Proxy 类中应用。在代码中，只需要将 flash_proxy 命名空间应用于扩展 Proxy 类的类。

flash_proxy 命名空间按照与下面类似的方法在 flash.utils 包中定义。

```
package flash.utils
{
    public namespace flash_proxy;
}
```

该命名空间将应用于 Proxy 类的方法，如下面摘自 Proxy 类的代码所示。

```
public class Proxy
{
    flash_proxy function callProperty(name:*, ... rest):*
    flash_proxy function deleteProperty(name:*):Boolean
    ...
}
```

如代码 4.16 所示，必须先导入 Proxy 类和 flash_proxy 命名空间，随后必须声明自己的类，以便对 Proxy 类进行扩展（如果是在严格模式下进行编译，则还必须添加 dynamic 修饰符）。在覆盖 callProperty()方法时，必须使用 flash_proxy 命名空间。

代码 4.16　扩展 Proxy 类

```
package
{
    import flash.utils.Proxy;
    import flash.utils.flash_proxy;
    dynamic class MyProxy extends Proxy
    {
        flash_proxy override function callProperty(name:*, ...rest):*
        {
            trace("方法调用被截取: " + name);
        }
    }
}
```

如果创建 MyProxy 类的一个实例，并调用一个未定义的方法（如在代码 4.17 中调用的 testing()方法），Proxy 对象将截获对该方法的调用，并执行覆盖后的 callProperty()方法内部的语句（在本例中为一个简单的 trace()语句）。

代码 4.17　创建 MyProxy 类的实例

```
var mySample:MyProxy = new MyProxy();
mySample.testing();
```

上述代码的输出结果如下：

```
方法调用被截取: testing
```

将 Proxy 类的方法放在 flash_proxy 命名空间中有两个好处。第一个好处是，在扩展 Proxy 类的任何类的公共接口中，拥有单独的命名空间可提高代码的可读性。（在 Proxy 类中大约有 12 个可以覆盖的方法，所有这些方法都不能被直接调用。将所有这些方法都放在公共命名空间中可能会引起混淆。）第二个好处是，当 Proxy 子类中包含名称与 Proxy 类方法的名称匹配的实例方法时，使用 flash_proxy 命名空间可避免名称冲突。例如，可能希望将自己的某个方法命名为 callProperty()。代码 4.18 是可接受的，因为所用的 callProperty()方法位于另一个命名空间中。

代码 4.18　添加自定义 callProperty()方法

```
dynamic class MyProxy extends Proxy
{
    public function callProperty() {}
    flash_proxy override function callProperty(name:*, ...rest):*
    {
        trace("method call intercepted: " + name);
    }
}
```

当希望以一种无法由 4 个访问控制修饰符（public、private、internal 和 protected）实现的方式，提供对方法或属性的访问时，命名空间也可能会非常有用。例如，可能有几个分散在多个包中的实用程序方法。希望这些方法对于所有包均可用，但是不希望这些方法成为公共方法。为此，可以创建一个新的命名空间，并将它用作自己的特殊访问控制修饰符。

下面的示例使用用户定义的命名空间，将两个位于不同包中的函数组合在一起。通过将它们组合到同一个命名空间中，可以通过一条 use namespace 语句，使这两个函数对于某个类或某个包均可见。

本示例使用 4 个文件来说明此方法。所有的文件都必须位于类路径中。第 1 个文件（myInternal.as）用来定义 myInternal 命名空间，如代码 4.19 所示。由于该文件位于名为 example 的包中，因此必须将该文件放在名为 example 的文件夹中。该命名空间标记为 public，因此可以导入到其他包中。

代码 4.19　定义命名空间：myInternal.as

```
// example 文件夹中的 myInternal.as
package example
{
    public namespace myInternal = "http://www.adobe.com/2006/actionscript/
    examples";
}
```

第 2 个文件（Utility.as）和第 3 个文件（Helper.as）定义的类中，应包含可供其他包使用的方法。Utility 类位于 example.alpha 包中，这意味着该文件应放在 example 文件夹下的 alpha 子文件夹中。Helper 类位于 example.beta 包中，这意味着该文件应放在 example 文件夹下的 beta 子文件夹中。这两个包（example.alpha 和 example.beta）在使用命名空间之前必须先导入它。Utility.as 文件的内容如代码 4.20 所示，Helper.as 文件的内容如代码 4.21 所示。

代码 4.20　使用自定义命名空间：Utility.as

```
// example/alpha 文件夹中的 Utility.as
package example.alpha
{
    import example.myInternal;

    public class Utility
    {
        private static var _taskCounter:int = 0;

        public static function someTask()
        {
            _taskCounter++;
        }

        myInternal static function get taskCounter():int
```

```
            {
                return _taskCounter;
            }
        }
    }
```

代码 4.21　使用自定义命名空间：Helper.as

```
    // example/beta 文件夹中的 Helper.as
    package example.beta
    {
        import example.myInternal;
        public class Helper
        {
            private static var _timeStamp:Date;
            public static function someTask()
            {
                timeStamp = new Date();
            }
            myInternal static function get lastCalled():Date
            {
                return _timeStamp;
            }
        }
    }
```

第 4 个文件（NamespaceUseCase.as）是主应用程序文件，应是 example 文件夹的同级，其内容如代码 4.22 所示。在 Flex Builder 3 中，将此文件用作 ActionScript 项目的可执行应用程序文件。NamespaceUseCase 类导入 myInternal 命名空间，并使用它来调用位于其他包中的两个静态方法。在本示例中，使用静态方法的目的仅在于简化代码。在 myInternal 命名空间中既可以放置静态方法也可以放置实例方法。

代码 4.22　创建主应用程序文件

```
    // Nam espaceUseCase.as
    package
    {
        import flash.display.Sprite;
        // 导入命名空间
        import example.myInternal;
        // 导入 Utility 类
        import example.alpha.Utility;
        // 导入 Helper 类
        import example.beta.Helper;
        public class NamespaceUseCase extends Sprite
        {
```

```
        public function NamespaceUseCase()
        {
            use namespace myInternal;
            Utility.someTask();
            Utility.someTask();
            trace(Utility.taskCounter);
            Helper.someTask();
            trace(Helper.lastCalled);
        }
    }
}
```

保存所有文件，调试该项目，输出结果如下所示。

```
2
Thu Oct 16 11:21:39 GMT+0800 2008
```

4.3　枚举类

枚举是用户创建的自定义数据类型，用于封装一小组值。ActionScript 3.0 并不支持具体的枚举工具，这与 C++使用 enum 关键字或 Java 使用 Enumeration 接口不一样。不过，可以使用类或静态常量创建枚举。例如，Flash Player API 中的 PrintJob 使用名为 PrintJobOrientation 的枚举，来存储由 landscape 和 portrait 组成的一组值，如代码 4.23 所示。

代码 4.23　创建枚举类

```
public final class PrintJobOrientation
{
    public static const LANDSCAPE:String = "landscape";
    public static const PORTRAIT:String = "portrait";
}
```

按照惯例，枚举类是使用 final 修饰符声明的，因为不需要扩展该类。该类仅由静态成员组成，这表示不创建该类的实例，而是直接通过类对象来访问枚举值，如代码 4.24 所示。

代码 4.24　使用枚举类

```
var pj:PrintJob = new PrintJob();
if(pj.start())
{
    if (pj.orientation == PrintJobOrientation.PORTRAIT)
    {
        ...
    }
    ...
}
```

Flash Player API 中的所有枚举类都只包含 String、int 或 uint 类型的变量。使用枚举而不使用字符串或数字值的好处是，使用枚举更易于发现字面错误。如果枚举名输入错误，ActionScript 编译器会生成一个错误。如果使用字面值，存在拼写错误或使用了错误数字时，编译器并不会报错。如果拼错了字符串字面值，编译器并不生成错误。

创建枚举的第二种方法还包括使用枚举的静态属性创建单独的类。这种方法的不同之处在于，每一个静态属性都包含一个类实例，而不是字符串或整数值。例如，代码 4.25 为一星期中的各天创建了一个枚举类。

代码 4.25　使用枚举的静态属性创建单独的类

```
public final class Day
{
    public static const MONDAY:Day = new Day();
    public static const TUESDAY:Day = new Day();
    public static const WEDNESDAY:Day = new Day();
    public static const THURSDAY:Day = new Day();
    public static const FRIDAY:Day = new Day();
    public static const SATURDAY:Day = new Day();
    public static const SUNDAY:Day = new Day();
}
```

Flash Player API 并不使用这种方法，但是许多开发人员都使用，他们更喜欢使用这种方法提供的改进类型检查功能。例如，返回枚举值的方法可将返回值限定为枚举数据类型。代码 4.26 不但显示了返回一星期中各天的函数，还显示了将枚举类型用作类型注释的函数调用。

代码 4.26　返回枚举值的方法

```
function getDay():Day
{
    var date:Date = new Date();
    var retDay:Day;
    switch (date.day)
    {
        case 0:
            retDay = Day.MONDAY;
            break;
        case 1:
            retDay = Day.TUESDAY;
            break;
        case 2:
            retDay = Day.WEDNESDAY;
            break;
        case 3:
            retDay = Day.THURSDAY;
            break;
```

```
            case 4:
                retDay = Day.FRIDAY;
                break;
            case 5:
                retDay = Day.SATURDAY;
                break;
            case 6:
                retDay = Day.SUNDAY;
                break;
        }
        return retDay;
    }
    var dayOfWeek:Day = getDay();
```

4.4 继承

继承是面向对象程序设计的主要特征之一，可以让开发人员重用以前开发过的代码，这样可以节省程序设计的时间。继承就是在类之间建立一种相交关系，使得新定义的派生类的实例可以继承已有的父类的特征和能力，而且可以加入新的特性或者是修改已有的特性建立起类的新层次。

4.4.1 继承概述

继承是一种代码重用的形式，允许程序员基于现有类开发新类。现有类通常称为父类或超类，新类通常称为子类。继承的主要优势是，允许重复使用父类中的代码，但不修改现有代码。此外，继承不要求改变其他类与父类交互的方式。使用继承可将该类视为一个集成模块，可使用其他属性或方法对它进行扩展。使用 extends 关键字指明类从另一类继承。

通过继承还可以在代码中实现多态。有一种方法在应用于不同数据类型时，会有不同行为，多态就是指对这样的方法应用同一个方法名的能力。名为 Shape 的父类就是一个简单的例子，该类有名为 Circle 和 Square 的两个子类。Shape 类定义了名为 area()的方法，该方法返回形状的面积。如果已实现多态，则可以对 Circle 和 Square 类型的对象调用 area()方法，然后执行正确的计算。使用继承能实现多态，实现的方式是，允许子类继承和重新定义或覆盖父类中的方法。在代码 4.27 中，Circle 和 Square 两个类重新定义了 area()方法。

代码 4.27　方法的继承

```
public class Shape
{
    public function area():Number
    {
        return NaN;
```

```
        }
    }
    public class Circle extends Shape
    {
        private var radius:Number = 1;
        override public function area():Number
        {
            return (Math.PI * (radius * radius));
        }
    }
    public class Square extends Shape
    {
        private var side:Number = 1;
        override public function area():Number
        {
            return (side * side);
        }
    }
    var cir:Circle = new Circle();
    trace(cir.area());
    var sq:Square = new Square();
    trace(sq.area());
```

代码 4.27 的输出结果如下所示。

```
3.141592653589793
1
```

因为每个类定义一个数据类型，所以使用继承会在父类和扩展父类的类之间创建一个特殊关系。子类保证拥有其父类的所有属性，这意味着子类的实例总是可以替换父类的实例。例如，如果方法定义了 Shape 类型的参数（parameter），由于 Circle 扩展了 Shape，因此 Circle 类型的参数（argument）是合法的，如下代码所示。

```
function draw(shapeToDraw:Shape) {}
var myCircle:Circle = new Circle();
draw(myCircle);
```

4.4.2　属性的继承

属性的继承是指，在由超类创建子类时，也将其属性传递给子类的性质。但是，并不是所有属性都能得到传递，这与在超类中定义的属性的性质有关，下面讲解影响属性能否继承的因素。

1．实例属性和继承

对于实例属性，无论是使用 var 还是使用 const 关键字定义的，只要在父类中未使用 private

修饰符声明该属性，这些属性都可以由子类继承。例如，Flash Player API 中的 Event 类具有很多子类，它们继承了所有事件对象共有的属性。

对于某些类型的事件，Event 类包含了定义事件所需的所有属性。下面的代码是从 Event 类中摘录的，显示由子类继承的某些属性。由于继承了属性，因此任何子类的实例都可以访问这些属性。

```
public class Event
{
    public function get type():String;
    public function get bubbles():Boolean;
    ...
}
```

其他类型的事件，需要特有属性，而 Event 类中没有提供。这些事件是使用 Event 类的子类定义的，所以可向 Event 子类中添加新属性。MouseEvent 类就是这样的一个子类，它可添加与鼠标移动或鼠标单击相关的事件的特有属性，如 mouseMove 和 click 事件。下面的代码是从 MouseEvent 类中摘录的，它说明了在子类中存在，但在父类中不存在的属性的定义：

```
public class MouseEvent extends Event
{
    public static const CLICK:String      = "click";
    public static const MOUSE_MOVE:String = "mouseMove";
    ...
}
```

2．访问控制说明符和继承

如果某一属性是用 public 修饰符声明的，则该属性对任何位置的代码都可见。这表示 public 关键字与 private、protected 和 internal 关键字不同，它对属性继承没有任何限制。

如果属性是使用 private 修饰符声明的，该属性只在定义该属性的类中可见，这表示它不能由任何子类继承。此行为与以前版本的 ActionScript 不同，在这些版本中，private 修饰符的行为更类似于 ActionScript 3.0 中的 protected 修饰符。

protected 修饰符指出某一属性不仅在定义该属性的类中可见，而且还在所有子类中可见。与 Java 编程语言中的 protected 修饰符不一样，ActionScript 3.0 中的 protected 修饰符并不使属性对同一包中的所有其他类可见。在 ActionScript 3.0 中，只有子类可以访问使用 protected 修饰符声明的属性。此外，protected 属性对子类可见，不管子类和父类是在同一包中，还是在不同包中。

要限制某一属性在定义该属性的包中的可见性，使用 internal 修饰符或者不使用任何访问控制修饰符。未指定访问控制修饰符时，应用的默认访问控制说明符是 internal 访问控制修饰符。标记为 internal 的属性将只由位于在同一包中的子类继承。

可以使用下面的示例，来查看每一个访问控制修饰符如何影响跨越包边界的继承。首先在 Flex Builder 3 中创建一个名为 AccessControl 的 ActionScript 项目，然后新建 foo 和 bar 两个文件夹，在 foo 文件夹中创建一个名为 Base.as 的类文件，在 bar 文件夹中创建一个名为

Extender.as 的类文件。

首先来看 Base.as 文件的内容，在 foo 包中创建了 Base 类。该类中有一个 public 修饰符定义的 str 属性，这是本实例中的超类，如代码 4.28 所示。

代码 4.28　定义超类

```
// Base.as 位于名为 foo 的文件夹中
package foo
{
    public class Base
    {
        public var str:String = "hello";
    }
}
```

然后来看 Extender.as 文件的内容，该文件在 bar 包中创建 Extender 类，该类继承于代码 4.28 中创建的 Base 类，如代码 4.29 所示。

代码 4.29　创建子类

```
// Extender.as 位于名为 bar 的文件夹中
package bar
{
    import foo.Base;
    public class Extender extends Base
    {
        public function getString():String {
            return str;
        }
    }
}
```

最后在主程序文件中，实例化 Extender 类，调用该实例的 getString()方法。getString()方法返回子类继承的 str 属性。本书中给出的是 public 修饰符定义的 str 属性，读者可以更改为 private、protected 或 internal，查看效果。AccessControl.as 文件的内容如代码 4.30 所示。

代码 4.30　主程序文件

```
// 名为 AccessControl.as 文件中的主应用程序类
package{
    import flash.display.Sprite;
    import bar.Extender;
    public class AccessControl extends Sprite
    {
        public function AccessControl()
        {
            var myExt:Extender = new Extender();
```

```
            trace(myExt.getString());
        }
    }
}
```


4.4.3 方法的继承和覆盖

子类在继承了超类的属性的同时，也继承了需要继承的方法。方法的继承与属性的继承比较相似，但是方法的覆盖是属性不能做到的。本节除了介绍方法的继承的相关知识，还要介绍方法的覆盖。

静态方法不能继承，也不能覆盖。但是，实例方法可由子类继承，也可覆盖，只要符合以下两个条件。

- 实例方法在父类中不是使用 final 关键字声明的。当 final 关键字与实例方法一起使用时，该关键字指明程序员的设计目的是要禁止子类覆盖方法。
- 实例方法在父类中不是使用 private 访问控制修饰符声明的。如果某个方法在父类中标记为 private，则在子类中定义同名方法时不能使用 override 关键字，因为父类方法在子类中不可见。

要覆盖符合这些条件的实例方法，子类中的方法定义必须使用 override 关键字，且必须在以下几个方面与方法的超类版本相匹配。

- 覆盖方法必须与父类方法具有相同级别的访问控制。标记为内部的方法与没有访问控制说明符的方法具有相同级别的访问控制。
- 覆盖方法必须与父类方法具有相同的参数个数。
- 覆盖方法参数必须与父类方法参数具有相同的数据类型注释。
- 覆盖方法必须与父类方法具有相同的返回类型。

覆盖方法中的参数名不必与父类中的参数名相匹配，只要参数个数和每个参数的数据类相匹配即可。

1. super 语句

覆盖方法时，程序员经常希望在要覆盖的超类方法的行为上添加行为，而不是完全替换该行为。这需要通过某种机制，来允许子类中的方法，调用它本身的超类版本。super 语句就提供了这样一种机制，其中包含对直接超类的引用。代码 4.31 中定义了名为 Base 的类（其中包含名为 thanks()的方法），还包含名为 Extender 的 Base 类的子类（用于覆盖 thanks()方法）。Extender.thanks()方法使用 super 语句调用 Base.thanks()。

代码 4.31 使用 super 语句

```
package {
    import flash.display.MovieClip;
    public class SuperExample extends MovieClip
    {
        public function SuperExample()
```

```
            {
                var myExt:Extender = new Extender()
                trace(myExt.thanks());
            }
        }
    }
    class Base {
        public function thanks():String
        {
            return "谢谢";
        }
    }
    class Extender extends Base
    {
        override public function thanks():String
        {
            return super.thanks() + "汇智科技";
        }
    }
```

代码 4.31 的输出结果如下所示。

```
谢谢汇智科技
```

2．覆盖 getter 和 setter

虽然不能覆盖超类中定义的属性，但是可以覆盖 getter 和 setter。例如，代码 4.32 用于覆盖在 Flash Player API 的 MovieClip 类中定义的名为 currentLabel 的 getter。

代码 4.32　覆盖 getter 实例

```
package
{
    import flash.display.MovieClip;
    public class OverrideExample extends MovieClip
    {
        public function OverrideExample()
        {
            trace(currentLabel)
        }
        override public function get currentLabel():String
        {
            var str:String = "Override: ";
            str += super.currentLabel;
            return str;
        }
    }
}
```

代码 4.32 的输出结果如下所示。

```
Override:OverrideExample
```

4.5 接口

在应用程序中，接口就是一种约定，使得实现接口的类或结构在形式上保持一致。接口是为继承而存在的，如果没有继承，接口就没有任何意义。使用接口的目的在于使应用程序代码更加清晰和条理化。本节将详细介绍接口。

接口是方法声明的集合，以使不相关的对象能够彼此通信。例如，Flash Player API 定义了 IEventDispatcher 接口，其中包含的方法声明，可供类用于处理事件对象。IEventDispatcher 接口建立了标准方法，供对象相互传递事件对象。以下代码显示了 IEventDispatcher 接口的定义。

```
public interface IEventDispatcher
{
    function addEventListener(type:String, listener:Function,
            useCapture:Boolean=false, priority:int=0,
            useWeakReference:Boolean = false):void;
    function removeEventListener(type:String, listener:Function,
            useCapture:Boolean=false):void;
    function dispatchEvent(event:Event):Boolean;
    function hasEventListener(type:String):Boolean;
    function willTrigger(type:String):Boolean;
}
```

了解接口，就必须了解定义接口中的方法与实现接口中的方法之间的区别。定义接口中的方法包括调用该方法必须的所有信息，例如方法名、所有参数和返回类型。实现接口中的方法不仅包括这些信息，而且还包括方法执行的语句。接口定义只包含方法接口，实现接口的类负责定义方法的实现。

在 Flash Player API 中，EventDispatcher 类通过定义所有 IeventDispatcher 接口方法并在每个方法中添加方法体来实现 IEventDispatcher 接口。以下代码摘录自 EventDispatcher 类的定义。

```
public class EventDispatcher implements IEventDispatcher
{
    function dispatchEvent(event:Event):Boolean
    {
        /* 实现语句 */
    }
    ...
}
```

IEventDispatcher 接口用作一个协议，EventDispatcher 实例通过该协议处理事件对象，然

后将事件对象传递到也实现了 IeventDispatcher 接口的其他对象。

另一种描述接口的方法是：接口定义了数据类型，就像类一样。因此，接口可以用作类型注释，也像类一样。作为数据类型，接口还可以与需要指定数据类型的运算符一起使用，如 is 和 as 运算符。但是与类不同的是，接口不可以实例化。这个区别使很多程序员认为接口是抽象的数据类型，类是具体的数据类型。

1. 定义接口

接口定义的结构类似于类定义的结构，只是接口只能包含方法但不能包含方法体。接口不能包含变量或常量，但是可以包含 getter 和 setter。要定义接口，需要使用 interface 关键字。例如，下面的接口 IExternalizable 是 Flash Player API 中 flash.utils 包的一部分。IExternalizable 接口定义一个用于对对象进行序列化的协议，这表示将对象转换为适合在设备上存储或通过网络传输的格式。

```
public interface IExternalizable
{
    function writeExternal(output:IDataOutput):void;
    function readExternal(input:IDataInput):void;
}
```

注意，IExternalizable 接口是使用 public 访问控制修饰符声明的。只能使用 public 或 internal 访问控制修饰符来定义接口。接口定义中的方法声明，不能包含任何访问控制修饰符。

Flash Player API 遵循一种约定，其中接口名以大写 I 开始，但是可以使用任何合法的标识符作为接口名。接口定义经常位于包的顶级。接口定义不能放在类定义或另一个接口定义中。

接口可扩展一个或多个其他接口。例如，下面的接口 IExample 扩展了 Iexternalizable 接口。

```
public interface IExample extends IExternalizable
{
    function extra():void;
}
```

实现 IExample 接口的所有类，不但必须包括 extra()方法的实现，还要包括从 IExternalizable 接口中继承的 writeExternal()和 readExternal()方法的实现。

2. 在类中实现接口

类是唯一的可实现接口的 ActionScript 3.0 语言元素。在类声明中使用 implements 关键字可实现一个或多个接口。代码 4.33 定义了两个接口 IAlpha 和 IBeta 以及实现这两个接口的类 Alpha。

代码 4.33　定义和实现接口

```
interface IAlpha
```

```
{
    function foo(str:String):String;
}
interface IBeta
{
    function bar():void;
}
class Alpha implements IAlpha, IBeta
{
    public function foo(param:String):String {}
    public function bar():void {}
}
```

在实现接口的类中，实现的方法必须符合以下几点。

使用 public 访问控制标识符。

使用与接口方法相同的名称。

拥有相同数量的参数，每一个参数的数据类型，都要与接口方法参数的数据类型相匹配。

使用相同的返回类型。

不过，在命名所实现方法的参数时，有一定的灵活性。虽然实现方法的参数数目和每个参数的数据类型，必须与接口方法的参数数目和数据类型相匹配，但参数名不需要匹配。

另外，使用默认参数值也具有一定的灵活性。接口定义可以包含使用默认参数值的函数声明。实现这种函数声明的方法必须采用默认参数值，默认参数值是与接口定义中指定的值具有相同数据类型的一个成员，但是实际值不一定匹配。例如，以下代码定义的接口中包含一个使用默认参数值 3 的方法。

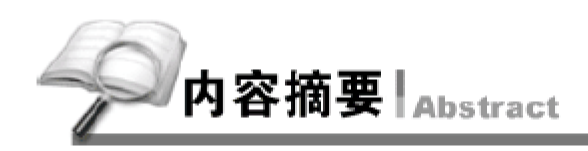
```
interface IGamma
{
    function doSomething(param:int = 3):void;
}
```

以下代码中，类定义实现 IGamma 接口，但使用不同的默认参数值：

```
class Gamma implements IGamma
{
    public function doSomething(param:int = 4):void {}
}
```

提供这种灵活性的原因是，实现接口的规则的设计目的是确保数据类型的兼容性，因此不必要求采用相同的参数名和默认参数值，就能实现目标。

第5章 ActionScript 3.0 中常用数据处理



内容摘要 Abstract

数据是程序的核心，对数据的处理是程序开发最常用的操作。在 ActionScript 3.0 中可以创建各种形式的数据，而且对不同形式的数据的处理也不一样。本章将介绍如何在 ActionScript 3.0 中处理函数、字符串、数组及日期和时间。



学习目标 Objective

- 掌握如何使用函数语句定义函数
- 熟悉函数表达式
- 掌握函数的调用
- 掌握函数的返回值、作用域及参数
- 掌握如何创建字符串
- 掌握 String 类的属性及常用方法
- 掌握如何创建索引数组和关联数组及对其的操作
- 熟悉如何创建多维数组及对其的操作
- 掌握如何创建 Date 对象及获取时间单位值
- 熟悉日期和时间运算
- 了解如何使用 Timer 类控制时间间隔

5.1 函数

函数是执行特定任务并可以在程序中重用的代码块。ActionScript 3.0 中有两类函数：方法和闭包函数。将函数称为方法还是闭包函数取决于定义函数的上下文。如果将函数定义为类的一部分或者将它附加到对象的实例，则该函数称为方法。如果以其他任何方式定义函数，则该函数称为闭包函数。

函数在 ActionScript 中始终扮演着极为重要的角色。例如，在 ActionScript 1.0 中，不存在 class 关键字，因此类由构造函数定义。尽管 class 关键字已经添加到了之后的 ActionScript 版本中，但是，如果想充分利用该语言所提供的功能，深入了解函数仍然十分重要。对于希望 ActionScript 中函数的行为与 C++或 Java 等语言中的函数的行为相似的程序员来说，这可能是一个挑战。尽管基本的函数定义和调用对有经验的程序员来说应该不是什么问题，但是仍需

要对 ActionScript 函数的一些更高级功能进行解释。

5.1.1 定义函数

在 ActionScript 3.0 中可通过两种方法来定义函数：使用函数语句和使用函数表达式。可以根据自己的编程风格（偏于静态还是偏于动态）来选择相应的方法。如果倾向于采用静态或严格模式的编程，则应使用函数语句来定义函数。如果有特定的需求，也可以使用函数表达式来定义函数，函数表达式更多地用在动态编程或标准模式编程中。

1. 函数语句

函数语句是在严格模式下定义函数的首选方法。函数语句以 function 关键字开头，后跟函数名、用小括号括起来的逗号分隔参数列表、冒号（:）、函数类型、用大括号括起来的函数体（即在调用函数时要执行的 ActionScript 代码）。例如，下面的代码创建一个函数，该函数的数据类型为 void。

```
function traceParameter(aParam:String):void
{
    trace(aParam);
}
```

2. 函数表达式

定义函数的第二种方法就是结合赋值语句使用的函数表达式，函数表达式有时也称为函数字面值或匿名函数。这是一种较为复杂的方法，仅在早期的 ActionScript 版本中广为使用。

带有函数表达式的赋值语句以 var 关键字开头，后跟函数名、冒号（:）、指示数据类型的 Function 类、赋值运算符（=）、function 关键字、用小括号括起来的逗号分隔参数列表、用大括号括起来的函数体。例如，下面的代码使用函数表达式来定义 traceParameter()函数。

```
var traceParameter:Function = function (aParam:String)
{
    trace(aParam);
};
```

函数表达式和函数语句的一个重要区别是，函数表达式是表达式，而不是语句。这意味着函数表达式不能独立存在，而函数语句则可以。函数表达式只能用作语句（通常是赋值语句）的一部分。代码 5.1 显示了一个赋予数组元素的函数表达式。

代码 5.1　赋予数组元素的函数表达式

```
var traceArray:Array = new Array();
traceArray[0] = function (aParam:String)
{
    trace(aParam);
};
traceArray[0]("hello");
```

代码 5.1 的执行结果如下所示。

```
hello
```

5.1.2　调用函数

可通过函数名称后跟小括号运算符（()）的标识来调用函数。要发送给函数的任何函数参数都括在小括号中。例如，贯穿于本书始末的 trace()函数，它是 Flash Player API 中的顶级函数。

```
trace("使用 trace 帮助调试脚本");
```

如果要调用没有参数的函数，则必须使用一对空的小括号。例如，可以使用没有参数的 Math.random()方法来生成一个随机数。

```
var randomNum:Number = Math.random();
```

5.1.3　函数的返回值

要从函数中返回值，就要使用 return 语句。该语句的参数类型要和函数的类型保持一致，否则会出现错误。例如，下面的函数，返回一个 int 类型的数据。

```
function doubleNum(baseNum:int):int
{
    return (baseNum * 2);
}
```

函数执行到 return 语句时，会终止该函数，而不会执行位于 return 语句下面的任何语句。如下所示的代码，将不会输出 baseNum*2 的执行结果。

```
function doubleNum(baseNum:int):int {
    return (baseNum * 2);
    trace(baseNum * 2); //不会执行这条 trace 语句
}
```

在严格模式下，如果指定返回类型，则必须返回相应类型的值。例如，下面的代码在严格模式下会生成错误，因为它不返回有效值。

```
function doubleNum(baseNum:int):int
{
    trace("after return");
}
```

5.1.4　函数的作用域

函数的作用域不但决定了可以在程序中的什么位置调用函数，而且还决定了函数可以访

问哪些定义。适用于变量标识符的作用域规则同样也适用于函数标识符。在全局作用域中声明的函数在整个代码中都可用。例如，ActionScript 3.0 包含可在代码中的任意位置使用的全局函数，如 isNaN()和 parseInt()。嵌套函数（即在另一个函数中声明的函数）可以用在声明它的函数中的任意位置。

1．作用域链

无论何时开始执行函数，都会创建许多对象和属性。首先，会创建一个称为激活对象的特殊对象，该对象用于存储在函数体内声明的参数和任何局部变量或函数。由于激活对象属于内部机制，因此无法直接访问它。接着，会创建一个作用域链，其中包含由 Flash Player 检查标识符声明的对象的有序列表。所执行的每个函数都有一个存储在内部属性中的作用域链。对于嵌套函数，作用域链始于其自己的激活对象，后跟其父函数的激活对象。作用域链以这种方式延伸，直到到达全局对象。全局对象是在 ActionScript 程序开始时创建的，其中包含所有的全局变量和函数。

2．闭包函数

闭包函数是一个对象，其中包含函数的快照及其词汇环境。函数的词汇环境包括函数作用域链中的所有变量、属性、方法和对象以及它们的值。无论何时在对象或类之外的位置执行函数，都会创建闭包函数。闭包函数保留定义它们的作用域，这样，在将函数作为参数或返回值传递给另一个作用域时，会产生有趣的结果。

例如，代码 5.2 创建两个函数：foo()（返回一个用来计算矩形面积的嵌套函数 rectArea()）和 bar()（调用 foo()并将返回的闭包函数存储在名为 myProduct 的变量中）。即使 bar()函数定义了自己的局部变量 x（值为 2），当调用闭包函数 myProduct()时，该闭包函数仍保留在函数 foo()中定义的变量 x（值为 40）。因此，bar()函数将返回值 160，而不是 8。

代码 5.2　将函数作为返回值传递给另一个作用域

```
function foo():Function
{
   var x:int = 40;
    // 定义闭包函数
   function rectArea(y:int):int
   {
      return x * y
   }
   return rectArea;
}
function bar():void
{
   var x:int = 2;
   var y:int = 4;
   var myProduct:Function = foo();
   // 调用闭包函数
```

```
        trace(myProduct(4));
    }
    bar(); // 160
```

方法的行为与闭包函数类似，因为方法也保留有关创建它们的词汇环境的信息。当方法提取自它的实例（这会创建绑定方法）时，此特征尤为突出。闭包函数与绑定方法之间的主要区别在于，绑定方法中 this 关键字的值始终引用它最初附加到的实例，而闭包函数中 this 关键字的值可以改变。

5.1.5　函数的参数

ActionScript 3.0 为函数参数提供了一些功能，这些功能对于那些刚接触 ActionScript 语言的程序员来说可能是很陌生的。尽管大多数程序员都应熟悉按值或按引用传递参数这一概念，但是很多人可能都对 arguments 对象和...(rest)参数感到很陌生。

1．按值或按引用传递参数

在许多编程语言中，一定要了解按值传递参数与按引用传递参数之间的区别，二者之间的区别会影响代码的设计方式。

按值传递意味着将参数的值复制到局部变量中以便在函数内使用，这种方式不会改变初始变量的值。按引用传递意味着将只传递对参数的引用，而不传递实际值。这种方式的传递不会创建实际参数的任何副本，而是会创建一个对变量的引用并将它作为参数传递，并且会将它赋给局部变量以便在函数内部使用。局部变量是对函数外部的变量的引用，它能够更改初始变量的值。

在 ActionScript 3.0 中，所有的参数均按引用传递，因为所有的值都存储为对象。但是，属于基本数据类型（包括 Boolean、Number、int、uint 和 String）的对象具有一些特殊运算符，这使它们可以像按值传递一样工作。

例如，代码 5.3 创建一个名为 passPrimitives()的函数，该函数定义了两个类型均为 int、名称分别为 xParam 和 yParam 的参数。这些参数与在 passPrimitives()函数体内声明的局部变量类似。当使用 xValue 和 yValue 参数调用函数时，xParam 和 yParam 参数将用对 int 对象的引用进行初始化，int 对象由 xValue 和 yValue 表示。因为参数是基元值，所以它们像按值传递一样工作。尽管 xParam 和 yParam 最初仅包含对 xValue 和 yValue 对象的引用，但是，对函数体内的变量的任何更改都会导致在内存中生成这些值的新副本。

代码 5.3　类似按值传递的函数

```
function passPrimitives(xParam:int, yParam:int):void
{
    xParam++;
    yParam++;
    trace(xParam, yParam);
}
var xValue:int = 10;
```

```
var yValue:int = 15;
trace(xValue, yValue);
passPrimitives(xValue, yValue);
trace(xValue, yValue);
```

代码 5.3 的输出结果如下所示。

```
10 15
11 16
10 15
```

在 passPrimitives()函数内部，xParam 和 yParam 的值递增，但这不会影响 xValue 和 yValue 的值，如第 1 条 trace 语句所示。即使参数的命名与 xValue 和 yValue 变量的命名完全相同，也是如此，因为函数内部的 xValue 和 yValue 将指向内存中的新位置，这些位置不同于函数外部同名的变量所在的位置。

其他所有对象（即不属于基本数据类型的对象）始终按引用传递，这样就可以更改初始变量的值。例如，代码 5.4 创建一个名为 objVar 的对象，该对象具有两个属性：x 和 y。该对象作为参数传递给 passByRef()函数。因为该对象不是基元类型，所以它不但按引用传递，而且还保持一个引用。这意味着对函数内部的参数的更改将会影响到函数外部的对象属性。

代码 5.4　对象作为参数，按引用传递

```
function passByRef(objParam:Object):void
{
    objParam.x++;
    objParam.y++;
    trace(objParam.x, objParam.y);
}
var objVar:Object = {x:10, y:15};
trace(objVar.x, objVar.y);
passByRef(objVar);
trace(objVar.x, objVar.y);
```

代码 5.4 的输出结果如下所示。

```
10 15
11 16
11 16
```

objParam 参数与全局 objVar 变量，引用相同的对象。正如代码 5.4 中的 trace 语句所跟踪到的一样，对 objParam 对象的 x 和 y 属性所做的更改将反映在 objVar 对象中。

2. 默认参数值

ActionScript 3.0 中新增了为函数声明默认参数值的功能。如果在调用具有默认参数值的函数时省略了具有默认值的参数，那么，将使用在函数定义中为该参数指定的默认值。所有具有默认值的参数都必须放在参数列表的末尾。指定为默认值的值必须是编译时常量。如果某

个参数存在默认值，则会有效地使该参数成为可选参数。没有默认值的参数被视为必须的参数。

例如，代码 5.5 创建一个具有 3 个参数的函数，其中的两个参数具有默认值。当仅用一个参数调用该函数时，将使用其他两个参数的默认值。

代码 5.5　使用参数的默认值

```
function defaultValues(x:int, y:int = 3, z:int = 5):void
{
    trace(x, y, z);
}
defaultValues(1); // 1 3 5
```

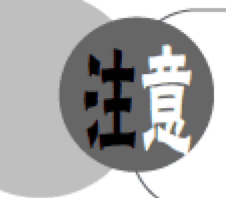
代码 5.5 的输出结果如下所示。

```
1 3 5
```

3．arguments 对象

在将参数传递给某个函数时，可以使用 arguments 对象来访问有关传递给该函数的参数的信息。arguments 对象的一些重要方面包括以下几点。

- arguments 对象是一个数组，其中包括传递给函数的所有参数。
- arguments.length 属性报告传递给函数的参数数量。
- arguments.callee 属性提供对函数本身的引用，该引用可用于递归调用函数表达式。



如果将任何参数命名为 arguments，或者使用...(rest)参数，则 arguments 对象不可用。

在 ActionScript 3.0 中，函数调用中所包括的参数的数量可以大于在函数定义中所指定的参数数量。但是，如果参数的数量小于必须参数的数量，在严格模式下将生成编译器错误。可以使用 arguments 对象的数组样式来访问传递给函数的任何参数，而无须考虑是否在函数定义中定义了该参数。代码 5.6 使用 arguments 数组及 arguments.length 属性来查看传递给 traceArgArray()函数的所有参数。

代码 5.6　使用 arguments 数组及 arguments.length 属性

```
function traceArgArray(x:int):void
{
    for (var i:uint = 0; i < arguments.length; i++)
    {
        trace(arguments[i]);
    }
}
traceArgArray(1, 2, 3);
```

代码 5.6 的输出结果如下所示。

```
1
2
3
```


arguments.callee 属性通常用在匿名函数中以创建递归。可以使用它来提高代码的灵活性。如果递归函数的名称在开发周期内的不同阶段会发生改变，而且使用的是 arguments.callee（而非函数名），则不必花费精力在函数体内更改递归调用。在代码 5.7 的函数表达式中，使用 arguments.callee 属性来启用递归。

代码 5.7　使用 arguments.callee 属性来启用递归

```
var factorial:Function = function (x:uint)
{
    if(x == 0)
    {
        return 1;
    }
    else
    {
        return (x * arguments.callee(x - 1));
    }
}
trace(factorial(5));
```

代码 5.7 的输出结果如下所示。

```
120
```

应避免将 arguments 字符串作为参数名，因为它将遮蔽 arguments 对象。例如，如果重写代码 5.6 中的 traceArgArray()函数，以便添加 arguments 参数，那么，函数体内对 arguments 的引用，将是该参数，而不是 arguments 对象。代码 5.8 不生成输出结果。

代码 5.8　使用 arguments 参数

```
function traceArgArray(x:int, arguments:int):void
{
    for (var i:uint = 0; i < arguments.length; i++)
    {
        trace(arguments[i]);
    }
}
traceArgArray(1, 2, 3);
```

在早期的 ActionScript 版本中，arguments 对象还包含一个名为 caller 的属性，该属性是对当前函数的引用。ActionScript 3.0 中没有 caller 属性，因此如果要通过引用调用函数，则可以更改调用函数，以使其传递一个额外的参数来引用它本身。

4．...(rest) 参数

ActionScript 3.0 中引入了一个名为...(rest)参数的新参数声明。此参数可用来指定一个数组参数，以接受任意多个以逗号分隔的参数。此参数可以拥有保留字以外的任意名称。此参数声明必须是最后一个指定的参数。使用此参数会使 arguments 对象变得不可用。尽管...(rest)参数提供了与 arguments 数组和 arguments.length 属性相同的功能，但是它不提供与 arguments.callee 类似的功能。使用...(rest)参数之前，应确保不需要使用 arguments.callee。

代码 5.9 中使用...(rest)参数（而非 arguments 对象）来重写代码 5.6 中的 traceArgArray()函数。

代码 5.9　使用...(rest)参数 1

```
function traceArgArray(... args):void
{
    for (var i:uint = 0; i < args.length; i++)
    {
        trace(args[i]);
    }
}
traceArgArray(1, 2, 3);
```

代码 5.9 的输出结果如下所示：

```
1
2
3
```

...(rest)参数还可与其他参数一起使用，前提是它是最后一个指定的参数。代码 5.10 修改 traceArgArray()函数，以便它的第一个参数 x 是 int 类型，第二个参数使用...(rest)参数。输出结果将忽略第一个值，因为第一个参数不再属于由...(rest)参数创建的数组。

代码 5.10　使用...(rest)参数 2

```
function traceArgArray(x: int, ... args)
{
    for (var i:uint = 0; i < args.length; i++)
    {
        trace(args[i]);
    }
}
traceArgArray(1, 2, 3);
```

代码 5.10 的输出结果如下所示。

```
2
3
```

5.2 字符串

在编程语言中，字符串是指一个文本值，即串在一起而组成单个值的一系列字母、数字或其他字符。在 ActionScript 中，可使用双引号或单引号将文本引起来以表示字符串值。

在 ActionScript 中处理一段文本时，都会用到字符串值。ActionScript 中的 String 类是一种可用来处理文本值的数据类型。String 实例通常用于很多其他 ActionScript 类中的属性、方法参数等。

5.2.1 创建字符串

在 ActionScript 3.0 中，String 类用于表示字符串（文本）数据。ActionScript 字符串既支持 ASCII 字符也支持 Unicode 字符。创建字符串的最简单方式是使用字符串文本。要声明字符串文本，可以使用双直引号（"）或单直引号（'）字符。例如，以下两个字符串是等效的。

```
var str1:String = "hello";
var str2:String = 'hello';
```

还可以使用 new 运算符来声明字符串，如下所示。

```
var str1:String = new String("hello");
var str2:String = new String(str1);
var str3:String = new String();       // str3 = ""
```

下面的两个字符串是等效的。

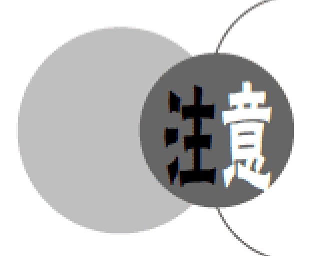
```
var str1:String = "hello";
var str2:String = new String("hello");
```

要在使用单直引号（'）分隔符定义的字符串文本内使用单直引号（'），则需要使用反斜杠转义符（\）。类似地，要在使用双直引号（"）分隔符定义的字符串文本内使用双直引号（"），也要使用反斜杠转义符（\）。下面的两个字符串是等效的。

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

可以根据字符串文本中存在的任何单直引号或双直引号来选择使用单直引号或双直引号，如下所示。

```
var str1:String = "ActionScript <span class='heavy'>3.0</span>";
var str2:String = '<item id="155">banana</item>';
```



ActionScript 可区分单直引号(')和左右单引号(‘或’)。对于双引号也同样如此，要使用直引号来分割字符串文本。在将文本从其他来源粘贴到 ActionScript 中时，确保使用正确的字符。

表 5-1 中列出了可以使用反斜杠转义符（\）在字符串文本中定义的特殊字符。

表 5-1　反斜杠转义符定义的特殊字符

转义序列	字符
\b	退格符
\f	换页符
\n	换行符
\r	回车符
\t	制表符
\u*nnnn*	Unicode 字符，字符代码由十六进制数字 *nnnn* 指定；例如，\u263a 为笑脸字符
\x*nn*	ASCII 字符，字符代码由十六进制数字 *nn* 指定
\'	单引号
\"	双引号
\\	单个反斜杠字符

除了上面自定义字符串外，还可以获取其他对象的字符串表示形式。ActionScript 3.0 中所有对象都提供了 toString()方法来实现此目的，如下所示的代码获得数字型数据的字符串形式。

```
var n:Number = 99.47;
var str:String = n.toString();
```

在使用+连接运算符连接 String 对象和不属于字符串的对象时，无须使用 toString()方法。有关字符串连接的详细信息，可参阅第 5.2.5 小节。对于给定对象，String()全局函数返回的值与调用该对象的 toString()方法返回的值相同。

5.2.2　String 类的属性和字符串中的字符

ActionScript 中的字符串，就是一个 String 类的实例，具有 String 类的属性和方法。本节将介绍，String 类的 length 属性及常用方法。

1．length 属性

每个字符串都有 length 属性，其值为一个整数，它指定 String 对象中的字符数。该属性的原始定义如下所示。

```
length:int  [read-only]
```

例如，下面的代码创建了一个字符串，通过 length 属性输出该字符串的字符数。

```
var str:String ="Adobe";
trace(str.length);
```

上述代码的输出结果如下所示。

```
5
```

所有字符串的索引都是从零开始的，所以任何字符串 x 的最后一个字符的索引都是

x.length-1。空字符串和 null 字符串的长度均为 0，如下例所示。

```
var str1:String = new String();
trace(str1.length);
str2:String = '';
trace(str2.length);
```

上述代码的输出结果如下所示。

```
0
0
```

2．字符串中的字符

上一节提到字符串索引，要想获得字符串指定位置的字符或 Unicode 字符代码，就要使用 charAt()方法或 charCodeAt()方法。这两个方法用来检查字符串中各个位置上的字符。

其中 charAt()方法，返回由参数 index 指定的位置处的字符。该方法的定义如下所示：

```
function charAt(index:Number = 0):String
```

该方法的参数 index:Number(default=0)是一个整数，指定字符在字符串中的位置。第一个字符由 0 指示，最后一个字符由 my_str.length–1 指示。

该方法的返回值为 String 类型数据，表示指定索引处的字符。如果指定的索引不在该字符串的索引范围内，即 index 不是从 0 到 string.length–1 之间的数字，则返回值为一个空字符串。

charCodeAt()方法，返回指定的 index 处字符的 Unicode 字符代码。该方法的定义如下所示：

```
function charCodeAt(index:Number = 0):Number
```

该方法的参数 index:Number(default=0)是一个整数，指定字符在字符串中的位置。第一个字符由 0 指示，最后一个字符由 my_str.length–1 指示。

该方法的返回值为 Number 类型数据，指定索引处字符的 Unicode 字符代码。如果索引不在此字符串的索引范围内，则返回 NaN。

此方法与 String.charAt()类似，所不同的是它返回的值是 16 位整型字符代码，而不是实际的字符。如代码 5.11 所示，分别使用 charAt()和 charCodeAt()方法，输出字符串的字符信息。

代码 5.11　使用 charAt()和 charCodeAt()方法

```
var str:String = "hello";
for (var:i = 0; i < str.length; i++)
{
    trace(str.charAt(i), "-", str.charCodeAt(i));
}
```

运行代码 5.11 时，会产生如下的输出结果。

```
h - 104
```

```
e - 101
l - 108
l - 108
o - 111
```

此外，还可以通过字符代码使用 fromCharCode()方法定义字符串，如下代码所示，创建"hello"字符串。


```
var myStr:String = String.fromCharCode(104,101,108,108,111);
```

3. 在大小写之间转换字符串

toLowerCase()方法和 toUpperCase()方法分别将字符串中的英文字母转换为小写和大写，如下面的代码所示。

```
var str:String = "Dr. Bob Roberts, #9."
trace(str.toLowerCase());
trace(str.toUpperCase());
```

上述代码的输出结果如下所示。

```
dr. bob roberts, #9.
DR. BOB ROBERTS, #9.
```

执行完这些方法后，源字符串 str 仍保持不变。要转换源字符串，需要使用下列代码。

```
str = str.toUpperCase();
```

这两种方法还可以处理扩展字符，而并不局限于 a～z 和 A～Z 的英文字母，如下代码所示。

```
var str:String = "Jose Barca";
trace(str.toUpperCase(), str.toLowerCase());
```

上述代码的输出结果如下所示。

```
JOSE BARCA jose barca
```

5.2.3　在字符串中查找子字符串和模式

子字符串是字符串内的字符序列。例如，字符串"汇智"具有如下子字符串：""、"汇"、"智"、"汇智"。模式是在 ActionScript 中通过字符串或正则表达式定义的。ActionScript 提供了在字符串中查找模式的方法，以及使用替换子字符串替换找到的匹配项的方法。

1. 通过字符位置查找子字符串

substr()和 substring()方法非常类似。两个方法都返回字符串的一个子字符串，并且两个方法都具有两个参数。在这两个方法中，第一个参数是子字符串的起始字符的位置。不过，在

substr()方法中，第二个参数是要返回的子字符串的长度，而在 substring()方法中，第二个参数是子字符串的结尾处字符的位置（该字符未包含在要返回的字符串中）。代码 5.12 显示了这两种方法之间的差别。

代码 5.12　使用 substr()和 substring()方法

```
var str:String = "我是汇智科技";
trace(str.substr(2,4));
trace(str.substring(2,4));
```

代码 5.12 的输出结果如下所示：

```
汇智科技
汇智
```

slice()方法的功能类似于 substring()方法。当指定两个非负整数作为参数时，其运行方式将完全一样。但是，slice()方法可以使用负整数作为参数，此时字符位置将从字符串末尾开始向前算起，如代码 5.13 所示。

代码 5.13　使用 slice()方法

```
var str:String = "我是汇智科技";
trace(str.slice(2,4));
trace(str.slice(-3,-1));
trace(str.slice(-6,str.length));
trace(str.slice(-5,-3));
```

代码 5.13 的输出结果如下所示：

```
汇智
智科
我是汇智科技
是汇
```

2. 查找匹配子字符串的字符位置

可以使用 indexOf()和 lastIndexOf()方法，在字符串内查找匹配的子字符串的索引位置。其中，indexOf()方法有两个参数，第一个参数为要搜索的子字符串，第二个参数为一个可选整数，指定搜索的起始索引，如代码 5.14 所示。

代码 5.14　使用 indexOf()方法

```
var str:String = "我是汇智科技，我是汇智人";
trace(str.indexOf("汇智"));
```

代码 5.14 的输出结果如下所示。

```
2
```

可以指定第二个参数以指出在字符串中开始进行搜索的起始索引位置，如代码 5.15 所示：

代码 5.15　使用 indexOf()方法指定起始索引位置

```
var str:String = "我是汇智科技，我是汇智人";
trace(str.indexOf("汇智",4));
```

代码 5.15 的输出结果如下所示。

```
9
```

lastIndexOf()方法从右向左搜索字符串，并返回在指定的索引之前，找到的最后一个匹配项的索引。此索引从零开始，这意味着第一个字符位于索引 0 处，最后一个字符位于 string.length–1 处。如果未找到匹配项，则该方法返回–1，如代码 5.16 所示。

代码 5.16　使用 lastIndexOf()方法

```
var str:String = "我是汇智科技，我是汇智人";
trace(str.lastIndexOf("汇智"));
```

代码 5.16 的输出结果如下所示。

```
9
```

如果为 lastIndexOf()方法提供了第二个参数，搜索将从字符串中的该索引位置反向（从右到左）进行，如代码 5.17 所示。

代码 5.17　使用 lastIndexOf()方法指定起始索引位置

```
var str:String = "我是汇智科技，我是汇智人";
trace(str.lastIndexOf("汇智",3));
```

代码 5.17 的输出结果如下所示。

```
2
```

3. 创建由分隔符分隔的子字符串数组

可使用 split()方法创建子字符串数组，该数组根据分隔符进行划分。split()方法的第一个参数为分隔符，例如，可以将用逗号分隔或空格分隔的字符串分为多个字符串。第二个参数是可选参数，该参数定义所返回数组的最大长度值。以下代码说明如何使用与字符（&）作为分隔符，将字符串分割为多个子字符串数组。

```
var queryStr:String = "汇智&科技&有限公司";
var params:Array = queryStr.split("&", 2); // params 为["汇智","科技"]
```

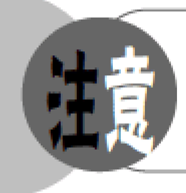
此外，还可以使用正则表达式作为分隔符。

```
var str:String = "汇智 科技\t 有限公司"
var a:Array = str.split(/\s+/); // a 为["汇智","科技","有限公司"]
```

4．查找匹配的子字符串

search()方法返回与给定模式相匹配的第一个子字符串的索引位置，该方法的定义如下所示：

```
function search(pattern:*):int
```



Search()方法搜索的字符串区分大小写。

该方法搜索指定的 pattern 并返回第一个匹配子字符串的索引。如果没有匹配的子字符串，则返回–1，如下面代码所示：

```
var str:String = "The more the better.";
trace(str.search("the"));
```

上述代码的输出结果如下所示。

```
9
```

还可以使用正则表达式定义要匹配的模式，search()方法仅查找一个匹配项并返回其起始索引位置，即便在正则表达式中设置了 g（全局）标志。如下代码使用正则表达式作为匹配模式，查找第一个匹配的位置。

```
var pattern:RegExp = /the/i;
var str:String = "The more the better.";
trace(str.search(pattern));
```

上述代码输出结果为 0，因为在正则表达式中设置了 i 标志，所以搜索时不区分大小写。匹配的字符串中第一个字符的索引位置为 0。

match()方法的工作方式与此类似。它搜索一个匹配的子字符串。但是，如果在正则表达式模式中使用了全局标志（如下例所示），match()方法将返回一个包含匹配子字符串的数组，如代码 5.18 所示。

代码 5.18　match()方法的使用

```
var str:Str ing = "www.itzcn.com,www.webzcn.org";
var pattern:RegExp = /\w*.\w*\.[org|com]+/g;
var results:Array = str.match(pattern);
```

代码 5.18 执行后，results 数组被设置为以下内容。

```
["www.itzcn.com"," www.webzcn.org"]
```

5.2.4　替换子字符串和模式

replace()方法对字符串匹配指定模式并返回一个新字符串，该方法的定义如下所示：

```
function replace(pattern:*, repl:Object):String
```

该方法的参数 pattern:*为要匹配的模式，可以为任何类型的对象，但通常是字符串或正则表达式。如果指定的 pattern 参数是除字符串或正则表达式以外的任何其他对象，将对该参数应用 toString()方法，并使用结果字符串作为 pattern 来执行 replace()方法。

repl:Object 参数通常是要插入的字符串，以用来替换匹配的内容。也可以指定一个函数作为此参数。如果指定一个函数，则将插入由该函数返回的字符串来替换匹配内容。代码 5.19 中，将正则表达式作为匹配模式，使用了 replace()方法替换子字符串。

代码 5.19　使用 replace 方法

```
var str:String = "She sells seashells by the seashore.";
var pattern:RegExp = /sh/gi;
trace(str.replace(pattern, "sch"));
```

在代码 5.19 中，在正则表达式中设置了 i(ignoreCase)标志，所以匹配的字符串是不区分大小写的；而且因为设置了 g(global)标志，所以会替换多个匹配项。运行后，输出结果如下所示。

```
sche sells seaschells by the seaschore.
```

可以在替换字符串中包括$替换代码，该替换代码有特殊的用途。表 5-2 中显示了$替换代码及所对应的被替换文本。

表 5-2　$替换代码及对应的被替换文本

$代码	被替换文本
$$	$
$&	匹配的子字符串
$`	字符串中位于匹配的子字符串前面的部分。请注意，此代码使用左侧直单引号字符（`），而不是直单引号字符（'）或左侧弯单引号字符（‘）
$'	字符串中位于匹配的子字符串后面的部分。请注意，此代码使用直单引号字符（'）
$*n*	第 *n* 个捕获的括号组匹配项，其中 *n* 是 1～9 之间的数字，而且$*n* 后面没有十进制数字
$*nn*	第 *nn* 个捕获的括号组匹配项，其中 *nn* 是一个十进制的两位数（01～99）。如果未定义第 *nn* 个捕获内容，则替换文本为空字符串

例如，下面说明了如何使用$2 和$1 替换代码，它们分别表示第一个和第二个匹配的捕获组。

```
var str:String = "汇智-科技";
var pattern:RegExp = /(\w+)-(\w+)/g;
trace(str.replace(pattern, "$2-$1"));
```

上述代码运行后，输出结果如下所示。

```
汇智-科技
```

指定一个函数作为 repl 时，replace()方法将以下参数传递给该函数。

- 字符串的匹配部分。
- 任何捕获到的括号组匹配项都将作为下一组参数提供。按这种方式传递的参数数目因括号匹配项的数目而异。可以通过检查函数代码中的 arguments.length-3 来确定括号匹配项的数目。
- 字符串中匹配开始的索引位置。
- 完整的字符串。

如代码 5.20 所示，将 replFN()函数作为 replace()方法的第二个参数。

代码 5.20　将函数作为 replace()方法的第二个参数

```
var str1:String = "abc12 def34";
var pattern:RegExp = /([a-z]+)([0-9]+)/;
var str2:String = str1.replace(pattern, replFN);
trace(str2);
function replFN():String {
    return arguments[2] + arguments[1];
}
```

上述代码中，对 replace()方法的调用，匹配两次正则表达式(/([a-z]([0-9]/g)。第一次，模式与子字符串"abc12"匹配，并将以下参数列表传递给该函数，{"abc12", "abc", "12", 0, "abc12 def34"}；第二次，模式与子字符串"def23"匹配，并将以下参数列表传递给该函数，{"def34", "def", "34", 6, "abc123 def34"}。最终输出的结果如下所示。

```
12abc 34def
```

5.2.5　字符串的连接与比较

前面几节介绍的大部分是对单一字符串的操作，本节介绍如何对多个字符串进行操作，包括连接和比较。

1. 字符串连接

字符串连接的含义是：将两个字符串按顺序合并为一个字符串。例如，可以使用+运算符来连接两个字符串。

```
var str1:String = "汇智";
var str2:String ="科技";
var str:String = str1 + str2;
```

上述代码执行后，变量 str 的值为"汇智科技"，还可以使用+=运算符得到相同的结果，如下代码所示。

```
var str:String = "汇智";
str +="科技";
```

如果使用+运算符（或+=运算符）对 String 对象和非字符串的对象进行运算，ActionScript

会自动将非字符串对象转换为 String 对象以计算该表达式。如下例所示，运行后，str 的值为 "Area=28.274333882308138"。

```
var str:String ="Area=";
var area:Number = Math.PI * Math.pow(3, 2);
str = str + area;
```

但是，可以使用括号进行分组，为+运算符提供运算的上下文，如下例所示。

```
trace("Total: $" + 4.55 + 1.45);
trace("Total: $" + (4.55 + 1.45));
```

上述代码的输出结果如下所示。

```
Total: $4.551.45
Total: $6
```

此外，String 类还包括 concat()方法，该方法在 String 对象末尾追加补充参数（如果需要，将它们转换为字符串）并返回结果字符串。源 String 对象的原始值保持不变。该方法的定义如下所示。

```
function concat(... args):String
```

下面的代码使用 concat()方法，运行后，str 的值为 www.itzcn.cn。

```
var str1:String = "www";
var str2:String = "itzcn";
var str3:String = "cn";
var str:String = str1.concat(".", str2, ".", str3);
```

2. 字符串比较

数字的比较，肯定都不陌生，例如 2>1 众所周知。字符串的比较相对来说比较少见，现实中，按字母顺序排序属于简单的字符串比较。由于计算机中储存的数据都是二进制数，也就为各种数据进行比较提供了可能。其中，字符串的比较是将字符串中字符的字符代码依次进行比较，直到比较出结果。

可以使用以下运算符比较字符串：<、<=、!=、==、=>和>。可以将这些运算符与条件语句（如 if 和 while）一起使用，如代码 5.21 所示。

代码 5.21　比较字符串

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2)
{
    trace("A < a, B < b, C < c, ...");
}
```

代码 5.21 将输出如下所示的结果。

```
A < a, B < b, C < c, ...
```

在将这些运算符用于字符串时，ActionScript 会使用字符串中每个字符的字符代码值从左到右比较各个字符，如代码 5.22 所示。

代码 5.22　演示从左到右比较各个字符

```
trace("A" < "a");
trace("Ab" < "Az");
trace("abc" < "abaz");
```

代码 5.22 运行后，输出结果如下所示：

```
true
true
false
```

使用==和!=运算符可比较两个字符串，也可以将字符串与其他类型的对象进行比较，如代码 5.23 所示。

代码 5.23　使用==运算符，将字符串与其他类型的对象进行比较

```
var str1:String = "1";
var str1b:String = "1";
var str2:String = "2";
trace(str1 == str1b);   // true
trace(str1 == str2);    // false
var total:uint = 1;
trace(str1 == total);   // true
```

代码 5.23 运行后，输出结果如下所示。

```
True
false
true
```

5.3　数组

使用数组可以在单数据结构中存储多个值。在 ActionScript 3.0 中可以使用简单的索引数组（使用固定有序的整数索引存储值），也可以使用复杂的关联数组（使用任意键存储值）。数组还可以是多维的，即包含本身是数组的元素。

5.3.1　数组简介

需要在编程中使用一组项目，而不是单个对象。例如，在音乐播放器应用程序中，可能

希望将等待播放的歌曲放在列表中。不希望必须为该列表中的每首歌曲创建单独的变量，而是希望将所有歌曲对象放在一个包中，因而能够将其作为一个组进行使用。

数组是一种编程元素，它用作一组项目的容器，如一组歌曲。通常，数组中的所有项目都是相同类的实例，但这在 ActionScript 中并不是必须的。数组中的各个项目称为数组的元素。可以将数组视为变量的文件柜，将变量作为元素添加到数组中，就像将文件夹放到文件柜中一样。当数组中包含一些元素后，可以将数组作为单个变量使用（就像将整个文件柜搬到其他地方一样），将这些变量作为组使用（就像逐个浏览文件夹以搜索一条信息一样），或者，可以分别访问它们（就像打开文件柜并选择单个文件夹一样）。

例如，要创建一个音乐播放器应用程序，用户可以在其中选择多首歌曲，并将这些歌曲添加到播放列表中。开发者可以在 ActionScript 代码中添加一个名为 addSongsToPlaylist()的方法，该方法接受单个数组作为参数。无论要将多少首歌曲（几首、很多首甚至只有一首）添加到列表中，都只需要调用一次 addSongsToPlaylist()方法，并将其传递给包含歌曲对象的数组。在 addSongsToPlaylist()方法中，可以使用循环来逐个访问数组元素（歌曲），并将歌曲添加到播放列表中。

最常见的 ActionScript 数组类型是索引数组，此数组将每个项目存储在编号位置（称为索引），可以使用该编号来访问项目，如地址。Array 类用于表示索引数组。索引数组可以很好地满足大多数编程的需要。索引数组的一个特殊用途是多维数组，此索引数组的元素也是索引数组（这些数组又包含其他元素）。另一种数组类型是关联数组，该数组使用字符串键来标识各个元素，而不是使用数字索引。最后，对于高级用户，ActionScript 3.0 还包括 Dictionary 类（表示字典），在此数组中，可以将任何类型的对象用作键来区分元素。

5.3.2　索引数组

索引数组存储一系列经过组织的单个或多个值，其中的每个值都可以通过使用一个无符号整数值进行访问。第一个索引始终是数字 0，且添加到数组中的每个后续元素的索引以 1 为增量递增。

索引数组使用无符号 32 位整数作为索引号。索引数组的最大索引号为 $2^{32}-1$，即 4 294 967 295。如果要创建的数组大小超过最大值，则会出现运行时错误。

数组元素的值可以为任意数据类型。ActionScript 3.0 不支持指定类型的数组概念，也就是说，不能指定数组的所有元素都属于特定数据类型。

可以调用 Array 类的构造函数或使用数组文本初始化数组，Array 类中还包含可用来修改索引数组的属性和方法。这些属性和方法几乎是专用于索引数组而非关联数组的。下面将讲述如何使用 Array 类创建和修改索引数组，首先讲的是如何创建数组。修改数组的方法分为 3 类，包括插入元素、删除元素和对数组进行排序。最后一类中的方法将现有数组当作只读数组，这些方法仅用于查询数组。所有的查询方法都返回新的数组，而非修改现有数组。

1．创建数组

Array 类构造函数的使用有 3 种方式。第一种，如果调用不带参数的构造函数，会得到空数组。可以使用 Array 类的 length 属性来验证数组是否包含元素。例如，以下代码调用不带参

数的 Array 类构造函数，输出结果为 0。

```
var names:Array = new Array();
trace(names.length);
```

第二种，如果将一个数字用作 Array 类构造函数的唯一参数，则会创建长度等于此数值的数组，并且每个元素的值都设置为 undefined。参数必须为介于值 0 和 4 294 967 295 之间的无符号整数。例如，以下代码调用带有一个数字参数的 Array 类构造函数。

```
var names:Array = new Array(3);
trace(names.length);
trace(names[0]);
trace(names[1]);
trace(names[2]);
```

上述代码的输出结果如下所示。

```
3
undefined
undefined
undefined
```

第三种，如果调用构造函数并传递一个元素列表作为参数，将创建具有与每个参数对应的元素的数组。以下代码将 3 个参数传递给 Array 类构造函数。

```
var names:Array = new Array("董红伟", "祝红涛", "李军华");
trace(names.length);    // 输出: 3
trace(names[0]);        // 输出: John
trace(names[1]);        // 输出: Jane
trace(names[2]);        // 输出: David
```

上述代码的输出结果如下所示。

```
3
董红伟
祝红涛
李军华
```

也可以创建具有数组文本或对象文本的数组。可以将数组文本直接分配给数组变量，如以下例所示。

```
var names:Array = ["董红伟", "祝红涛", "李军华"];
```

2. 插入数组元素

可以使用 Array 类的 3 种方法（push()、unshift()和 splice()）将元素插入数组。push()方法用于在数组末尾添加一个或多个元素。换言之，使用 push()方法在数组中插入的最后一个元素将具有最大索引号。unshift()方法用于在数组开头插入一个或多个元素，并且始终在索引号 0

处插入。splice()方法用于在数组中的指定索引处插入任意数目的项目。

代码 5.24 中对所有的 3 种方法进行了说明。它创建一个名为 planets 的数组，以便按照距离太阳的远近顺序存储各个行星的名称。首先，调用 push()方法以添加初始项 Mars。接着，调用 unshift()方法在数组开头插入项 Mercury。最后，调用 splice()方法在 Mercury 之后和 Mars 之前插入项 Venus 和 Earth。传递给 splice()的第一个参数是整数 1，它用于指示从索引 1 处开始插入。传递给 splice()的第二个参数是整数 0，它表示不应删除任何项。传递给 splice()的第三和第四个参数 Venus 和 Earth 为要插入的项。

代码 5.24　使用 Array 类的 3 种插入方法

```
var planets:Array = new Array();
// 数组内容: Mars
planets.push("Mars");
// 数组内容: Mercury,Mars
planets.unshift("Mercury");
// 数组内容: Mercury,Venus,Earth,Mars
planets.splice(1, 0, "Venus", "Earth");
trace(planets);
```

代码 5.24 的输出结果如下所示：

```
Mercury,Venus,Earth,Mars
```

push()和 unshift()方法均返回一个无符号整数，它们表示修改后的数组长度。在用于插入元素时，splice()方法返回空数组，这看上去也许有点奇怪，但考虑到 splice()方法的多用途性，便会觉得它很有意义。通过使用 splice()方法，不仅可以将元素插入到数组中，而且还可以从数组中删除元素。用于删除元素时，splice()方法将返回包含被删除元素的数组。

3．删除数组元素

可以使用 Array 类的 3 种方法（pop()、shift()和 splice()）从数组中删除元素。pop()方法用于从数组末尾删除一个元素。换言之，它将删除位于最大索引号处的元素。shift()方法用于从数组开头删除一个元素，也就是说，它始终删除索引号 0 处的元素。splice()方法既可用来插入元素，又可以删除任意数目的元素，其操作的起始位置位于由传送到此方法的第一个参数指定的索引号处。

代码 5.25 中使用所有 3 种方法从数组中删除元素。它创建一个名为 oceans 的数组，以便存储海洋的名称。数组中的某些名称为河流的名称而非海洋的名称，因此需要将其删除。

代码 5.25　使用 Array 类的 3 种删除方法

```
var oceans:Array = ["珠江", "北冰洋", "黄河", "长江", "太平洋", "淮河"];
//替换黄河和长江
oceans.splice(2, 2, "大西洋", "印度洋");
//删除淮河
oceans.pop();
```

```
//删除珠江
oceans.shift();
trace(oceans);
```


首先，使用 splice()方法删除“黄河”和“长江”，并插入大西洋和印度洋。传递给 splice()的第一个参数是整数 2，它表示应从列表中的第 3 项（即索引 2 处）开始执行操作。第二个参数 2 表示应删除两项。其余两个参数“大西洋”和“印度洋”是要在索引 2 处插入的值。

然后，使用 pop()方法删除数组中的最后一个元素“淮河”。最后，使用 shift()方法删除数组中的第 1 项“珠江”。

代码 5.25 的输出结果如下所示。

```
北冰洋,大西洋,印度洋,太平洋
```

pop()和 shift()方法均返回已删除的项。由于数组可以包含任意数据类型的值，因而返回值的数据类型为 Object。splice()方法将返回包含被删除项的数组。可以以更改 oceans 数组示例，以使 splice()调用将被删除项的数组分配给新的数组变量，部分代码如下所示：

```
var rivers:Array = oceans.splice(2, 2, "大西洋", "印度洋");
trace(rivers);
```

上述代码的输出结果如下所示。

```
黄河,长江
```

delete 运算符用于将数组元素的值设置为 undefined，但它不会从数组中删除元素。例如，代码 5.26 在 oceans 数组的第 3 个元素上使用 delete 运算符，但此数组的长度仍然为 4。

代码 5.26　使用 Array 类的 delete 运算符

```
var oceans:Array = ["北冰洋", "大西洋", "印度洋", "太平洋"];
delete oceans[2];
trace(oceans);
trace(oceans[2]);
trace(oceans.length);
```

代码 5.26 的输出结果如下所示。

```
"北冰洋", "大西洋", "印度洋", "太平洋"
undefined
4
```

可以使用数组的 length 属性截断数组。如果将数组的 length 属性设置为小于数组当前长度的值，则会截断数组，在索引号高于 length 的新值减 1 处所存储的任何元素都将被删除。例如，如果 oceans 数组的排序是将所有有效项放在数组的开始处，则可以使用 length 属性删除数组末尾的项，如代码 5.27 所示。

代码 5.27　使用 Array 类的 length 属性截断数组

```
var oceans:Array = ["北冰洋", "大西洋", "印度洋", "太平洋", "黄河", "长江"];
```

```
oceans.length = 4;
trace(oceans);
```

代码 5.27 的输出结果如下所示。

```
"北冰洋", "大西洋", "印度洋", "太平洋"
```


4．对数组排序

可以使用 3 种方法（reverse()、sort()和 sortOn()）通过排序或反向排序来更改数组的顺序。所有这些方法都用来修改现有数组。reverse()方法用于按照以下方式更改数组的顺序：最后一个元素变为第一个元素，倒数第二个元素变为第二个元素，依次类推。sort()方法可用来按照多种预定义的方式对数组进行排序，甚至可用来创建自定义排序算法。sortOn()方法可用来对对象的索引数组进行排序，这些对象具有一个或多个可用作排序键的公共属性。

reverse()方法不带参数，也不返回值，但可以将数组从当前顺序切换为相反顺序。代码 5.28 颠倒了 oceans 数组中列出的海洋顺序。

代码 5.28　reverse()方法颠倒数组中元素的顺序

```
var oceans:Array = ["北冰洋", "大西洋", "印度洋", "太平洋"];
oceans.reverse();
trace(oceans);
```

代码 5.28 的输出结果如下所示。

```
"太平洋","印度洋","大西洋","北冰洋"
```

sort()方法按照默认排序顺序重新安排数组中的元素。默认排序顺序具有以下特征。

- 排序区分大小写，也就是说大写字符优先于小写字符。例如，字母 D 优先于字母 b。
- 排序按照升序进行，也就是说低位字符代码（例如 A）优先于高位字符代码（例如 B）。
- 排序将相同的值互邻放置，并且不区分顺序。
- 排序基于字符串，也就是说，在比较元素之前，先将其转换为字符串（例如，10 优先于 3，因为相对于字符串"3"而言，字符串"1"具有低位字符代码）。

也许需要不区分大小写或者按照降序对数组进行排序，或者数组中包含数字，从而需要按照数字顺序而非字母顺序进行排序。sort()方法具有 options 参数，可通过该参数改变默认排序顺序的各个特征。options 是由 Array 类中的一组静态常量定义的，如表 5-3 所示。

表 5-3　参数 options 的值与实现功能

参数 options 的值	实现功能
Array.CASEINSENSITIVE	可使排序不区分大小写。例如，小写字母 b 优先于大写字母 D
Array.DESCENDING	用于颠倒默认的升序排序。例如，字母 B 优先于字母 A
Array.UNIQUESORT	如果发现两个相同的值，此选项将导致排序中止
Array.NUMERIC	这会导致排序按照数字顺序进行，比方说 3 优先于 10

代码 5.29 重点说明了这些选项中的某些选项。它创建一个名为 poets 的数组，并使用几种

不同的选项对其进行排序。

代码 5.29　使用 sort()方法，不同参数 options 的值

```
var poets:Array = ["Blake", "cummings", "Angelou", "Dante"];
// 默认排序
poets.sort();
trace(poets);
poets.sort(Array.CASEINSENSITIVE);
trace(poets);
poets.sort(Array.DESCENDING);
trace(poets);
// 使用两个选项
poets.sort(Array.DESCENDING | Array.CASEINSENSITIVE);
trace(poets);
```

代码 5.29 的输出结果如下所示。

```
Angelou,Blake,Dante,cummings
Angelou,Blake,cummings,Dante
cummings,Dante,Blake,Angelou
Dante,cummings,Blake,Angelou
```

也可以编写自定义排序函数，然后将其作为参数传递给 sort()方法。例如，如果有一个名称列表，其中每个列表元素都包含一个人的全名，但现在要按照姓来对列表排序，则必须使用自定义排序函数解析每个元素，然后使用排序函数中的姓对列表排序。代码 5.30 说明如何使用自定义函数作为参数，传递给 Array.sort()方法来完成上述工作。

代码 5.30　使用自定义函数作为参数，传递给 Array.sort()方法

```
var names:Array = new Array("John Q. Smith", "Jane Doe", "Mike Jones");
function orderLastName(a, b):int
{
    var lastName:RegExp = /\b\S+$/;
    var name1 = a.match(lastName);
    var name2 = b.match(lastName);
    if (name1 < name2)
    {
        return -1;
    }
    else if (name1 > name2)
    {
        return 1;
    }
    else
    {
        return 0;
```

```
        }
    }
    trace(names); // 输出: John Q. Smith,Jane Doe,Mike Jones
    names.sort(orderLastName);
    trace(names); // 输出: Jane Doe,Mike Jones,John Q. Smith
```

代码 5.30 的输出结果如下所示。

```
John Q. Smith,Jane Doe,Mike Jones
Jane Doe,Mike Jones,John Q. Smith
```

自定义排序函数 orderLastName()使用正则表达式从每个元素中提取姓，以用于比较操作。针对 names 数组调用 sort()方法时，函数标识符 orderLastName 用作唯一的参数。排序函数接受两个参数 a 和 b，因为它每次对两个数组元素进行操作。排序函数的返回值指示应如何对元素排序。

- 返回值为–1 表示第一个参数 a 优先于第二个参数 b。
- 返回值为 1 表示第二个参数 b 优先于第一个参数 a。
- 返回值为 0 表示元素具有相同的排序优先级。

sortOn()方法是为包含对象元素的索引数组设计的。这些对象应至少具有一个可用作排序键的公共属性。如果将 sortOn()方法用于任何其他类型的数组，则会产生意外结果。

代码 5.31 修改 poets 数组，以使每个元素均为对象而非字符串。每个对象既包含诗人的姓又包含诗人的出生年份。

代码 5.31　修改 poets 数组每个元素均为对象

```
var poets:Array = new Array();
poets.push({name:"Angelou", born:"1928"});
poets.push({name:"Blake", born:"1757"});
poets.push({name:"cummings", born:"1894"});
poets.push({name:"Dante", born:"1265"});
poets.push({name:"Wang", born:"701"});
```

可以使用 sortOn()方法，按照 born 属性对数组进行排序。sortOn()方法定义两个参数 fieldName 和 options，必须将 fieldName 参数指定为字符串。代码 5.32 中，使用两个参数 born 和 Array.NUMERIC 来调用 sortOn()。Array.NUMERIC 参数用于确保按照数字顺序进行排序，而不是按照字母顺序。即使所有数字具有相同的数位，这也是一种很好的做法，因为当后来在数组中添加较少数位或较多数位的数字时，它会确保排序继续进行。

代码 5.32　调用 sortOn()方法

```
poets.sortOn("born", Array.NUMERIC);
for (var i:int = 0; i < poets.length; ++i)
{
    trace(poets[i].name, poets[i].born);
}
```

代码 5.32 的输出结果如下所示。

```
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
```


通常，sort()和 sortOn()方法用来修改数组。如果要对数组排序而又不修改现有数组，要将 Array.RETURNINDEXEDARRAY 常量作为 options 参数的一部分进行传递。此选项将指示方法返回反映排序的新数组同时保留原始数组。方法返回的数组为由反映新排序顺序的索引号组成的简单数组，不包含原始数组的任何元素。例如，如果要根据出生年份对 poets 数组排序，同时不对数组进行修改，可在传递的 options 参数中包括 Array.RETURNINDEXEDARRAY 常量。

代码 5.33 将返回的索引信息存储在名为 indices 的数组中，然后使用 indices 数组和未修改的 poets 数组按出生年份顺序输出诗人。

代码 5.33　输出返回的索引信息

```
var indices:Array;
indices = poets.sortOn("born", Array.NUMERIC | Array.RETURNINDEXEDARRAY);
for (var i:int = 0; i < indices.length; ++i)
{
    var index:int = indices[i];
    trace(poets[index].name, poets[index].born);
}
```

代码 5.33 的输出结果如下所示。

```
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
```

5．查询数组

Array 类中的其余 4 种方法 concat()、join()、slice()和 toString()用于查询数组中的信息，而不修改数组。concat()和 slice()方法返回新数组，而 join()和 toString()方法返回字符串。concat()方法将新数组和元素列表作为参数，并将其与现有数组结合起来创建新数组。slice()方法具有两个名为 startIndex 和 endIndex 的参数，并返回一个新数组，它包含从现有数组分离出来的元素副本。分离从 startIndex 处的元素开始，到 endIndex 处的前一个元素结束。值得强调的是，endIndex 处的元素不包括在返回值中。

代码 5.34 通过 concat()和 slice()方法，使用其他数组的元素创建一个新数组。

代码 5.34　使用 concat()和 slice()方法

```
var array1:Array = ["alpha", "beta"];
var array2:Array = array1.concat("gamma", "delta");
trace(array2);
var array3:Array = array1.concat(array2);
trace(array3);
var array4:Array = array3.slice(2,5);
trace(array4);
```

代码 5.34 的输出结果如下所示。

```
alpha,beta,gamma,delta
alpha,beta,alpha,beta,gamma,delta
alpha,beta,gamma
```

可以使用 join()和 toString()方法查询数组，并将其内容作为字符串返回。如果 join()方法没有使用参数，则这两个方法的行为相同，它们都返回包含数组中所有元素列表（以逗号分隔）的字符串。与 toString()方法不同，join()方法接受名为 delimiter 的参数，可以使用此参数，选择要用作返回字符串中各个元素之间分隔符的符号。

代码 5.35 创建名为 rivers 的数组，并调用 join()和 toString()方法以便按字符串形式返回数组中的值。toString()方法用于返回以逗号分隔的值，而 join()方法用于返回以+字符分隔的值。

代码 5.35　使用 join()和 toString()方法

```
var rivers:Array = ["长江", "黄河", "淮河", "珠江"];
var riverCSV:String = rivers.toString();
trace(riverCSV);
var riverPSV:String = rivers.join("+");
trace(riverPSV);
```

代码 5.35 的输出结果如下所示。

```
长江,黄河,淮河,珠江
长江+黄河+淮河+珠江
```

5.3.3　关联数组

关联数组有时候也称为哈希或映射，它使用键而非数字索引来组织存储的值。关联数组中的每个键都是用于访问一个存储值的唯一字符串。关联数组为 Object 类的实例，也就是说每个键都与一个属性名称对应。关联数组是键和值对的无序集合。在代码中，不应期望关联数组的键按特定的顺序排列。本节说明如何创建使用字符串和对象作为键的数组。

1．具有字符串键的关联数组

在 ActionScript 3.0 中有两种创建关联数组的方法。第一种方法是使用 Object 类构造函数，

它的优点是可以使用对象文本初始化数组。Object 类的实例（也称作通用对象）在功能上等同于关联数组。通用对象的每个属性名称都用作键，提供对存储的值的访问。

代码 5.36 创建一个名为 monitorInfo 的关联数组，并使用对象文本初始化具有两个键和值对的数组。

代码 5.36　使用对象文本创建关联数组

```
var monitorInfo:Object = {type:"纯平", resolution:"1600 x 1200"};
trace(monitorInfo["type"], monitorInfo["resolution"]);
```

代码 5.36 的输出结果如下所示。

```
纯平 1600×1200
```

如果在声明数组时不需要初始化，可以使用 Object 类构造函数创建数组，代码如下所示。

```
var monitorInfo:Object = new Object();
```

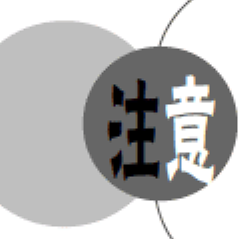
使用对象文本或 Object 类构造函数创建数组后，可以使用括号运算符（[]）或点运算符（.）在数组中添加值。代码 5.37 将两个新值添加到 monitorArray 中。

代码 5.37　将两个新值添加到数组

```
monitorInfo["aspect ratio"] = "16:10";
monitorInfo.colors = "16.7 million";
trace(monitorInfo["aspect ratio"], monitorInfo.colors);
```

代码 5.37 的输出结果如下所示。

```
16:10 16.7 million
```



名为 aspect ratio 的键包含空格字符。也就是说，空格字符可以与括号运算符一起使用，但试图与点运算符一起使用时会生成一个错误。不建议在键名称中使用空格。

第二种关联数组的创建方法是使用 Array 类构造函数，然后使用括号运算符（[]）或点运算符（.）将键和值对添加到数组中。如果将关联数组声明为 Array 类型，则将无法使用对象文本初始化该数组。代码 5.38 使用 Array 构造函数创建一个名为 monitorInfo 的关联数组，并添加一个名为 type 的键和一个名为 resolution 的键以及它们的值。

代码 5.38　使用 Array 类构造函数创建关联数组

```
var monitorInfo:Array = new Array();
monitorInfo["type"] = "纯平";
monitorInfo["resolution"] = "1600 x 1200";
trace(monitorInfo["type"], monitorInfo["resolution"]);
```

代码 5.38 的输出结果如下所示。

纯平 1600×1200

使用 Array 类构造函数创建关联数组没有什么优势。即使使用 Array 类构造函数或 Array 数据类型，也不能将 Array 类的 Array.length 属性或任何方法用于关联数组。最好将 Array 类构造函数用于创建索引数组。

2．具有对象键的关联数组

可以使用 Dictionary 类创建使用对象而非字符串作为键的关联数组。这样的数组有时候也称作字典、哈希或映射。例如，考虑这样一个应用程序，可根据 Sprite 对象与特定容器的关联确定 Sprite 对象的位置。可以使用 Dictionary 对象，将每个 Sprite 对象映射到一个容器。

以下代码创建 3 个用作 Dictionary 对象的键的 Sprite 对象实例。它为每个键分配了值 GroupA 或 GroupB。值可以是任意数据类型，但在此示例中，GroupA 和 GroupB 均为 Object 类的实例。然后，可以使用属性访问运算符（[]）访问与每个键关联的值，如代码 5.39 所示。

代码 5.39　使用 Dictionary 类创建关联数组

```
import flash.display.Sprite;
import flash.utils.Dictionary;
var groupMap:Dictionary = new Dictionary();
// 要用作键的对象
var spr1:Sprite = new Sprite();
var spr2:Sprite = new Sprite();
var spr3:Sprite = new Sprite();
// 要用作值的对象
var groupA:Object = new Object();
var groupB:Object = new Object();
// 在字典中创建新的键-值对。
groupMap[spr1] = groupA;
groupMap[spr2] = groupB;
groupMap[spr3] = groupB;
if (groupMap[spr1] == groupA)
{
    trace("spr1 is in groupA");
}
if (groupMap[spr2] == groupB)
{
    trace("spr2 is in groupB");
}
if (groupMap[spr3] == groupB)
{
    trace("spr3 is in groupB");
}
```

代码 5.39 的输出结果如下所示。

```
spr1 is in groupA
spr2 is in groupB
spr3 is in groupB
```

3. 使用对象键循环访问

可以使用 for…in 循环或 for each…in 循环来循环访问 Dictionary 对象的内容。for…in 循环用来基于键进行循环访问，而 for each…in 循环用来基于与每个键关联的值进行循环访问。

可以使用 for…in 循环直接访问 Dictionary 对象的对象键，还可以使用属性访问运算符([])访问 Dictionary 对象的值。代码 5.40 使用代码 5.39 中的 groupMap 字典来说明如何使用 for…in 循环来循环访问 Dictionary 对象。

代码 5.40　使用 for…in 循环访问 Dictionary 对象的内容

```
for (var key:Object in groupMap)
{
    trace(key, groupMap[key]);
}
```

代码 5.40 的输出结果如下所示。

```
[object Sprite] [object Object]
[object Sprite] [object Object]
[object Sprite] [object Object]
```

也可以使用 for each…in 循环直接访问 Dictionary 对象的值。代码 5.41 也使用 groupMap 字典来说明如何使用 for each…in 循环来循环访问 Dictionary 对象。

代码 5.41　使用 for each…in 循环访问 Dictionary 对象的内容

```
for each (var item:Object in groupMap)
{
    trace(item);
}
```

代码 5.41 的输出结果如下所示。

```
[object Object]
[object Object]
[object Object]
```

5.3.4　多维数组

多维数组将其他数组作为其元素。例如，考虑一个任务列表，它存储为字符串索引数组，代码如下所示。

```
var tasks:Array = ["上课", "写作业"];
```

如果要将一周中每天的任务存储为一个单独的列表，可以创建一个多维数组，一周中的每天使用一个元素。每个元素包含一个与 tasks 数组类似的索引数组，而该索引数组存储任务列表。在多维数组中，可以使用任意组合的索引数组和关联数组。

1．两个索引数组

使用两个索引数组时，可以将结果呈现为表或电子表格。第一个数组的元素表示表的行，第二个数组的元素表示表的列。

例如，代码 5.42 中，多维数组使用两个索引数组跟踪一周中每一天的任务列表。第 1 个数组 masterTaskList 是使用 Array 类构造函数创建的。此数组中的各个元素分别表示一周中的各天，其中索引 0 表示星期一，索引 6 表示星期日。可将这些元素当成是表的行。可通过为 masterTaskList 数组中创建的 7 个元素中的每个元素分配数组文本来创建每 1 天的任务列表。这些数组文本表示表的列。

代码 5.42　使用两个索引数组创建多维数组

```
var masterTaskList:Array = new Array();
masterTaskList[0] = ["上课", "写作业"];
masterTaskList[1] = ["上课", "打扫卫生", "写作业"];
masterTaskList[2] = ["考试", "温习"];
masterTaskList[3] = ["上课", "体育锻炼"];
masterTaskList[4] = ["上课", "音乐欣赏"];
masterTaskList[5] = ["上课", "美术欣赏"];
masterTaskList[6] = ["逛街", "游玩"];
```

可以使用括号记号访问任意任务列表中的单个项。第 1 组括号表示一周的某一天，第 2 组括号表示这一天的任务列表。例如，要检索星期三的列表中的第 2 项任务，请首先使用表示星期三的索引 2，然后使用表示列表中的第二项任务的索引 1。

```
trace(masterTaskList[2][1]);
```

上述代码的输出结果如下所示。

```
温习
```

2．具有索引数组的关联数组

要使单个数组的访问更加方便，可以使用关联数组表示一周的各天，并使用索引数组表示任务列表。通过使用关联数组可以在引用一周中特定的一天时使用点语法，但要访问关联数组的每个元素，还需额外进行运行时处理。代码 5.43 使用关联数组作为任务列表的基础，并使用键和值对来表示一周中的每一天。

代码 5.43　使用关联数组创建多维数组

```
var masterTaskList:Object = new Object();
masterTaskList["Monday"] = ["上课", "写作业"];
```

```
masterTaskList["Tuesday"] = ["上课", "打扫卫生", "写作业"];
masterTaskList["Wednesday"] = ["考试", "温习"];
masterTaskList["Thursday"] = ["上课", "体育锻炼"];
masterTaskList["Friday"] = ["上课", "音乐欣赏"];
masterTaskList["Saturday"] = ["上课", "美术欣赏"];
masterTaskList["Sunday"] = ["逛街", "游玩"];
trace(masterTaskList.Wednesday[1]);
trace(masterTaskList.Sunday[0]);
```


点语法通过避免使用多组括号改善了代码的可读性。代码 5.43 的输出结果如下所示。

```
温习
逛街
```

5.3.5　克隆数组

Array 类不具有复制数组的内置方法，可以通过调用不带参数的 concat()或 slice()方法来创建数组的浅副本。在浅副本中，如果原始数组具有对象元素，则仅复制指向对象的引用而非对象本身。与原始数组一样，浅副本也指向相同的对象。对对象所做的任何更改都会在两个数组中反映出来。

在深副本中，将复制原始数组中的所有对象，从而使新数组和原始数组指向不同的对象。深度复制需要多行代码，通常需要创建函数。可以将此类函数作为全局实用程序函数或 Array 子类的方法来进行创建。

代码 5.44 定义一个名为 clone()的函数以执行深度复制。其算法采用了一般的 Java 编程技巧。此函数创建深副本的方法是：将数组序列化为 ByteArray 类的实例，然后将此数组读回到新数组中。此函数接受对象，因此既可以将此函数用于索引数组，又可以将其用于关联数组，如下所示。

代码 5.44　执行深度复制的 clone()函数

```
import flash.utils.ByteArray;
function clone(source:Object):*
{
    var myBA:ByteArray = new ByteArray();
    myBA.writeObject(source);
    myBA.position = 0;
    return(myBA.readObject());
}
```

5.4　日期和时间

时间可能并不代表一切，但它在软件应用程序中通常是一个关键要素。ActionScript 3.0

提供了多种强大的手段来管理日历日期、时间和时间间隔。以下两个主类提供了大部分的计时功能：Date 类和 flash.utils 包中的新 Timer 类。

日期和时间是在 ActionScript 程序中使用的一种常见信息类型。例如，了解当前星期值，或测量用户在特定屏幕上花费多少时间，并且还可能会执行很多其他操作。在 ActionScript 中，可以使用 Date 类来表示某一时刻，其中包含日期和时间信息。Date 实例中包含各个日期和时间单位的值，其中包括年、月、日、星期、小时、分钟、秒、毫秒以及时区。本节将介绍如何创建 Date 对象，通过该对象获取时间单位值和执行运算或转换。对于更高级的用法，ActionScript 还包括 Timer 类，可以使用该类在一定延迟后执行动作，或按重复间隔执行动作。

5.4.1　创建 Date 对象

ActionScript 3.0 的所有日历日期和时间管理函数都集中在顶级 Date 类中。Date 类包含一些方法和属性，这些方法和属性能够按照通用协调时间（UTC）或特定于时区的本地时间来处理日期和时间。UTC 是一种标准时间定义，它实质上与格林尼治标准时间（GMT）相同。

Date 类是所有核心类中构造函数方法形式最为多变的类之一。可以用如下 4 种方式来构造 Date 类。

第一，如果未给定参数，则 Date()构造函数将按照用户所在时区的本地时间返回包含当前日期和时间的 Date 对象。下面是一个实例。

```
var now:Date = new Date();
```

第二，如果仅给定了一个数字参数，则 Date()构造函数将其视为自 1970 年 1 月 1 日以来经过的毫秒数，并且返回对应的 Date 对象。注意，传入的值将被视为自 1970 年 1 月 1 日（UTC 时间）以来经过的毫秒数。如果仅使用一个毫秒参数来创建新的 Date 对象，则应考虑到当地时间和 UTC 之间的时区差异。以下语句创建一个设置为 1970 年 1 月 1 日午夜（UTC 时间）的 Date 对象。

```
var millisecondsPerDay:int = 1000 * 60 * 60 * 24;
// 获取一个表示自起始日期 1970 年 1 月 1 日后又过了一天时间的 Date 对象
var startTime:Date = new Date(millisecondsPerDay);
```

第三，可以将多个数值参数传递给 Date()构造函数。该构造函数将这些参数分别视为年、月、日、小时、分钟、秒和毫秒，并将返回一个对应的 Date 对象。假定这些输入参数采用的是本地时间而不是 UTC。以下语句获取一个设置为 2000 年 1 月 1 日开始的午夜（本地时间）的 Date 对象：

```
var millenium:Date = new Date(2000, 0, 1, 0, 0, 0, 0);
```

第四，可以将单个字符串参数传递给 Date()构造函数。该构造函数将尝试把字符串解析为日期或时间部分，然后返回对应的 Date 对象。如果使用此方法，最好将 Date()构造函数包含在 try..catch 块中以捕获任何解析错误。以下语句使用字符串值初始化一个新的 Date 对象。

```
var nextDay:Date = new Date("Mon May 1 2008 11:30:00 AM");
```

如果 Date()构造函数无法成功解析该字符串参数，它将不会引发异常。但是，所得到的 Date 对象将包含一个无效的日期值。

5.4.2 获取时间单位值

可以使用 Date 类的属性或方法从 Date 对象中提取各种时间单位的值。表 5-4 中列出了 Date 类的属性及说明，这些属性提供了 Date 对象中的一个时间单位的值。

表 5-4 Date 类的属性及说明

属性	说明
fullYear 属性	按照本地时间返回 Date 对象中的完整年份值（一个 4 位数，例如 2008）
month 属性	以数字格式表示，分别以 0 到 11 表示 1 月到 12 月
date 属性	表示月中某一天的日历数字，范围从 1 到 31
day 属性	以数字格式表示一周中的某一天，其中 0 表示星期日
hours 属性	按照本地时间返回 Date 对象中一天的小时值（0 到 23 之间的一个整数）
minutes 属性	按照本地时间返回 Date 对象的分钟值（0 到 59 之间的一个整数）部分
seconds 属性	按照本地时间返回 Date 对象的秒值（0 到 59 之间的一个整数）
milliseconds 属性	按照本地时间返回 Date 对象中的毫秒值（0 到 999 之间的一个整数）部分

实际上，Date 类提供了获取这些值的多种方式。例如，可以用 4 种不同方式获取 Date 对象的月份值。

- month 属性
- getMonth()方法
- monthUTC 属性
- getMonthUTC()方法

所有 4 种方式实质上具有同等的效率，因此可以任意使用一种最适合应用程序的方法。上文列出的属性，其值都是有效值，系统会根据时间自动改变。例如，milliseconds 属性永远不会大于 999，因为当它达到 1000 时，秒钟值就会增加 1 并且 milliseconds 属性会重置为 0。

如果要获得 Date 对象自 1970 年 1 月 1 日(UTC)起所经过毫秒数的值，可以使用 getTime()方法。通过使用与其相对应的 setTime()方法，可以更改自 1970 年 1 月 1 日（UTC）起经过的毫秒数来更改现有 Date 对象的值。

5.4.3 执行日期和时间运算

可以使用 Date 类对日期和时间执行加法和减法运算。日期值在内部以毫秒的形式保存，因此应将其他值转换成毫秒，然后再将它们与 Date 对象进行加减。

如果应用程序将执行大量的日期和时间运算，可能会发现创建常量来保存常见时间单位值（以毫秒的形式）非常有用，如下所示。

```
public static const millisecondsPerMinute:int = 1000 * 60;
public static const millisecondsPerHour:int = 1000 * 60 * 60;
```

```
public static const millisecondsPerDay:int = 1000 * 60 * 60 * 24;
```

现在，可以方便地使用标准时间单位来执行日期运算。下列代码使用 getTime()和 setTime()方法将日期值设置为当前时间一个小时后的时间。

```
var oneHourFromNow:Date = new Date();
oneHourFromNow.setTime(oneHourFromNow.getTime() + millisecondsPerHour);
```

设置日期值的另一种方式，是仅使用一个毫秒参数创建新的 Date 对象。例如，下列代码将一个日期加上 30 天以计算另一个日期。

```
// 将日期设置为今天的日期
var invoiceDate:Date = new Date();
// 加上 30 天以获得到期日期
var dueDate:Date = new Date(invoiceDate.getTime() + (30 * millisecondsPerDay));
```

接着，将 millisecondsPerDay 常量乘以 30 以表示 30 天的时间，并将得到的结果，与 invoiceDate 的 getTime()方法返回的值相加，并将其用于设置 dueDate 值。

在需要将日期从一种时区转换成另一种时区时，使用日期和时间运算十分方便。也可以使用 getTimezoneOffset()方法，该方法返回的值表示 Date 对象的时区与 UTC 之间相差的分钟数。此方法之所以返回以分钟为单位的值，是因为并不是所有时区之间都正好相差一个小时，有些时区与邻近的时区仅相差半个小时。

代码 5.45 使用时区偏移量将日期从本地时间转换成 UTC。该示例首先以毫秒为单位计算时区值，然后按照该量调整 Date 值。

代码 5.45　将日期从本地时间转换成 UTC

```
// 按本地时间创建 Date
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");
// 通过加上或减去时区偏移量，将 Date 转换为 UTC
var offsetMilliseconds:Number = nextDay.getTimezoneOffset() * 60 * 1000;
nextDay.setTime(nextDay.getTime() + offsetMilliseconds);
```

5.4.4　控制时间间隔

使用 Adobe Flash CS3 Professional 开发应用程序时，可以访问时间轴，这样可以稳定且逐帧地完成该应用程序。但在 Adobe Flex Builder 3 中，必须依靠其他计时机制。

1. 循环与计时器之比较

在某些编程语言中，必须使用循环语句（如 for 或 do…while）来设计计时方案。通常，循环语句会以本地计算机所允许的速度尽可能快地执行，这表明应用程序在某些计算机上的运行速度较快，而在其他计算机上则较慢。如果应用程序需要一致的计时间隔，则需要将其与实际的日历或时钟时间联系在一起。而许多应用程序（如游戏、动画和实时控制器）需要在不同计算机上均能保持一致的、规则的时间驱动计时机制。

ActionScript 3.0 的 Timer 类提供了一个功能强大的解决方案。使用 ActionScript 3.0 事件模型，Timer 类在每次达到指定的时间间隔时都会调度计时器事件。

2. Timer 类

在 ActionScript 3.0 中处理计时函数的首选方式是使用 Timer 类（flash.utils.Timer），可以使用它在每次达到间隔时调度事件。

要启动计时器，请先创建 Timer 类的实例，并告诉它每隔多长时间生成一次计时器事件以及在停止前生成多少次事件。

例如，下列代码创建一个每秒调度一个事件且持续 60 秒的 Timer 实例。

```
var oneMinuteTimer:Timer = new Timer(1000, 60);
```

Timer 对象在每次达到指定的间隔时，都会调用 TimerEvent 对象。TimerEvent 对象的事件类型是 timer（由常量 TimerEvent.TIMER 定义）。TimerEvent 对象包含的属性与标准 Event 对象包含的属性相同。

如果将 Timer 实例设置为固定的间隔数，则在达到最后一次间隔时，它还会调用 timerComplete 事件（由常量 TimerEvent.TIMER_COMPLETE 定义）。

代码 5.46 是一个用来展示 Timer 类实际操作的小示例应用程序，该程序需要在 Adobe Flex Builder 3 中新建一个 ActionScript Project。

代码 5.46　使用 Timer 类

```
package
{
    import flash.display.Sprite;
    import flash.events.TimerEvent;
    import flash.utils.Timer;
    public class ShortTimer extends Sprite
    {
        public function ShortTimer()
        {
            // 创建一个新的 5 秒的 Timer
            var minuteTimer:Timer = new Timer(1000, 5);
            // 为间隔和完成事件指定侦听器
            minuteTimer.addEventListener(TimerEvent.TIMER, onTick);
            minuteTimer.addEventListener(TimerEvent.TIMER_COMPLETE,
            onTimerComplete);
            // 启动计时器计时
            minuteTimer.start();
        }
        public function onTick(event:TimerEvent):void
        {
            // 显示到目前为止的时间计数
            // 该事件的目标是 Timer 实例本身
```

```
                trace("tick" + event.target.currentCount);
            }
            public function onTimerComplete(event:TimerEvent):void
            {
                trace("时间到! ");
            }
        }
    }
```

在代码 5.46 中创建 ShortTimer 类时，首先创建一个用于每秒计时 1 次，并持续 5 秒的 Timer 实例。然后，它将两个侦听器添加到计时器：一个用于侦听每次计时，另一个用于侦听 timerComplete 事件。

接着，启动计数器计时，以 1 秒钟的间隔执行 onTick()方法。onTick()方法只显示当前的时间计数。5 秒钟后，执行 onTimerComplete()方法，告诉时间已到。

运行该示例时，会看到下列行，以每秒一行的速度显示在控制台中。

```
tick 1
tick 2
tick 3
tick 4
tick 5
时间到!
```

3．flash.utils 包中的计时函数

ActionScript 3.0 包含许多与 ActionScript 2.0 提供的计时函数类似的计时函数。这些函数是作为 flash.utils 包中的包级别函数提供的，这些函数的工作方式与 ActionScript 2.0 中的完全相同。表 5-5 列出了 flash.utils 包中的计时函数及其说明。

表 5-5　flash.utils 包中的计时函数及其说明

函数	说明
clearInterval(id:uint):void	取消指定的 setInterval()调用
clearTimeout(id:uint):void	取消指定的 setTimeout()调用
getTimer():int	返回自 Adobe Flash Player 被初始化以来经过的毫秒数
setInterval(closure:Function, delay:Number, ... arguments):uint	以指定的间隔（以毫秒为单位）运行函数
setTimeout(closure:Function, delay:Number, ... arguments):uint	在指定的延迟（以毫秒为单位）后运行指定的函数

这些函数仍保留在 ActionScript 3.0 中以实现向后兼容。Adobe 不建议在新的 ActionScript 3.0 应用程序中使用这些函数。通常，在应用程序中使用 Timer 类会更容易且更有效。

第 2 篇　ASP.NET 编程篇

第6章 ASP.NET 的简单应用


内容摘要 | Abstract

ASP.NET 因其简单、快捷、高效的特性，越来越得到广大用户的青睐，特别是当 ASP.NET 3.5 推出之后。它是基于.NET 平台的一个革命性突破，而且解决了过去 Web 开发技术中存在的各种不足和局限，具有运行速度快、安全等特点。本章将简单介绍 ASP.NET 的运行环境、C#语言基础语法以及如何配置 ASP.NET 应用程序等内容。


学习目标 | Objective

- 了解.NET Framework 3.5
- 了解 ASP.NET 3.5 开发环境
- 掌握 C#控制语句
- 熟悉面向对象实现
- 了解 C#中结构、枚举、数组和集合的使用方法
- 了解 Web.config 配置文件
- 掌握如何在 Flex 中生成配置文件

6.1 ASP.NET 3.5 概述

ASP.NET 是 Microsoft .NET Framework 的一部分，是一种可以在高度分布的 Internet 环境中简化应用程序开发的计算环境。ASP.NET 提供了为建立和部署企业级 Web 应用程序所必须的服务，并且为能够面向任何浏览器或设备的更安全、更强的可升级性、更稳定的应用程序提供新的编程模型和基础结构。

6.1.1 .NET Framework 3.5 简介

.NET Framework 是支持生成和运行下一代应用程序和 XML Web Services 的内部 Windows 组件。.NET Framework 旨在实现下列目标。

- 提供一个一致的面向对象的编程环境，而无论对象代码是在本地存储和执行，还是在本地执行但在 Internet 上分布，或者是在远程执行。
- 提供一个将软件部署和版本控制冲突最小化的代码执行环境。

- 提供一个可提高代码（包括由未知的或不完全受信任的第三方创建的代码）执行安全性的代码执行环境。
- 提供一个可消除脚本环境或解释环境的性能问题的代码执行环境。
- 使开发人员的经验在面对类型大不相同的应用程序（如基于 Windows 的应用程序和基于 Web 的应用程序）时保持一致。
- 按照工业标准生成所有通信，以确保基于.NET Framework 的代码可与任何其他代码集成。

1. .NET Framework 组件

.NET Framework 主要有两个组件：公共语言运行库和.NET Framework 类库。公共语言运行库是.NET Framework 的基础。读者可以将运行库看作一个在执行时管理代码的代理，它提供内存管理、线程管理和远程处理等核心服务，并且还强制实施严格的类型安全以及可提高安全性和可靠性的其他形式的代码准确性。事实上，代码管理的概念是运行库的基本原则。以运行库为目标的代码称为托管代码，而不以运行库为目标的代码称为非托管代码。

.NET Framework 的另一个主要组件是类库，它是一个综合性的面向对象的可重用类型集合，读者可以使用它开发多种应用程序，这些应用程序包括传统的命令行或图形用户界面（GUI）应用程序，也包括基于 ASP.NET 所提供的最新的应用程序（如 Web 窗体和 XML Web Services）。

在图 6-1 中所示的.NET Framework 平台上显示了公共运行时和类库与应用程序以及与整个系统之间的关系。

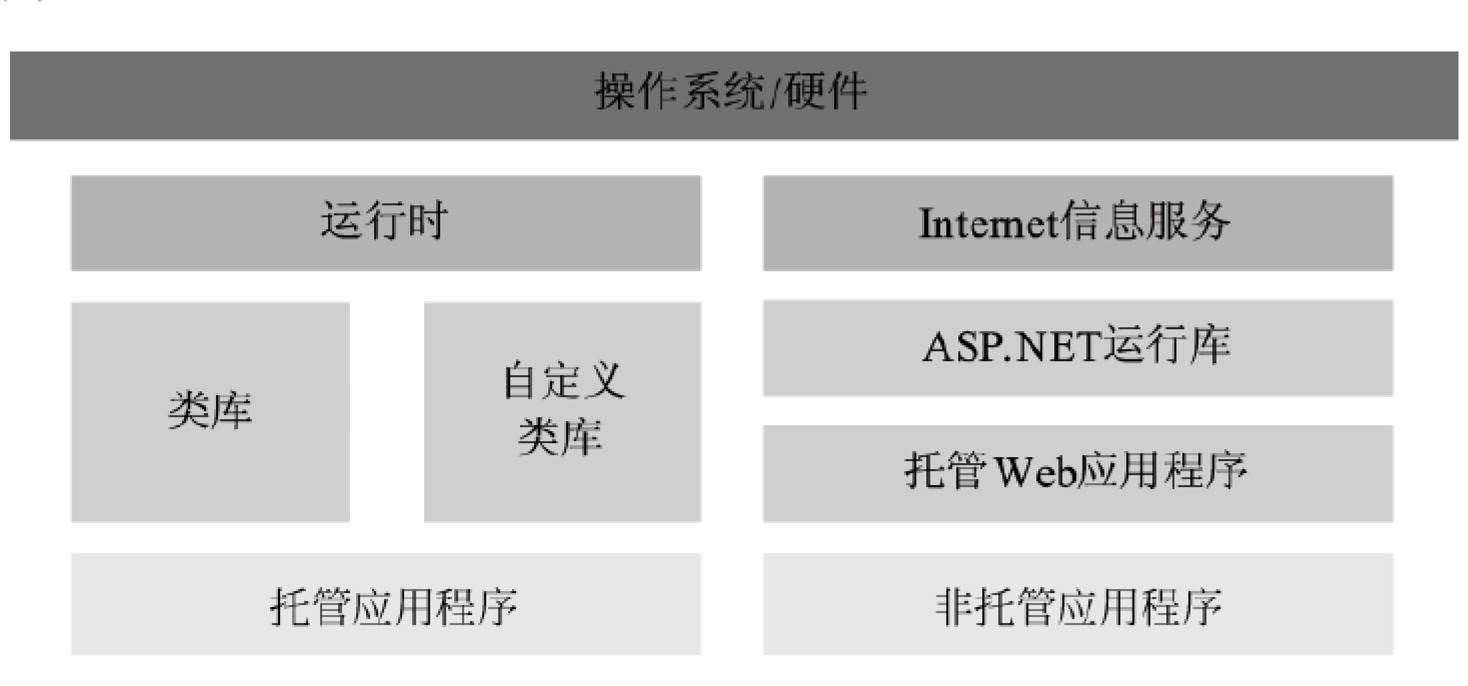


图 6-1 .NET Framework 平台

2. .NET Framework 3.5 简介

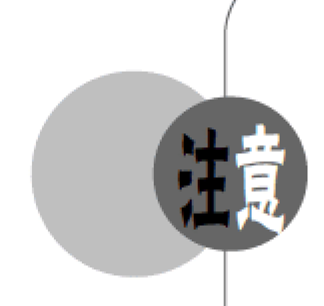
.NET Framework 3.5 版以.NET Framework 2.0 版和.NET Framework 3.0 版为基础，包括.NET Framework 2.0 和 3.0 版的 Service Pack。主要包括如下的组件。

- .NET Framework 2.0。
- .NET Framework 2.0 Service Pack 1，它更新包含在.NET Framework 2.0 中的程序集。
- .NET Framework 3.0，它使用.NET Framework 2.0 或.NET Framework 2.0 SP1（如果已安装）中存在的程序集，并且包含.NET Framework 3.0 中引入的技术所必需的程序集。例如，Windows Presentation Foundation（WPF）所必需的 PresentationFramework.dll 和

PresentationCore.dll 都随.NET Framework 3.0 一起安装。

- .NET Framework 3.0 Service Pack 1，它更新在.NET Framework 3.0 中引入的程序集。
- 一些新程序集为.NET Framework 2.0 和 3.0 提供附加功能，同时还提供.NET Framework 3.5 中新采用的技术。

如果在计算机上安装 .NET Framework 3.5 时缺少上述任何组件，则这些组件会自动安装。应用程序无论针对的是.NET Framework 2.0、3.0 还是 3.5 版，都使用相同的程序集。例如，对于使用 WPF 并针对.NET Framework 3.0 的应用程序，其所使用的 mscorlib 程序集实例与使用 Windows 窗体并针对.NET Framework 2.0 的应用程序相同。如果.NET Framework 2.0 SP1 已安装在计算机上，则 mscorlib.dll 会更新，并且两个应用程序将都使用 mscorlib.dll 的更新版本。



.NET Framework 2.0、3.0 和 3.5 版之间的关系不同于 1.0、1.1 和 2.0 版之间的关系。.NET Framework 1.0、1.1 和 2.0 版彼此完全独立，对于其中任何一个版本来说，无论计算机上是否存在其他版本，自己都可以存在于该计算机上。当 1.0、1.1 和 2.0 版位于同一台计算机上时，每个版本都有自己的公共语言运行库、类库和编译器等。应用程序可以选择是针对 1.0、1.1 还是 2.0 版的。

3. .NET Framework 3.5 的重要新功能

.NET Framework 3.5 为 2.0 和 3.0 中的技术引入了新功能，并以新程序集的形式引入了其他技术。下列技术是随.NET Framework 3.5 引入的技术。

- **LINQ**

LINQ（Language Integrate Query，语言集成查询）是 Visual Studio 2008 和.NET Framework 3.5 中的新功能。LINQ 将强大的查询功能扩展到 C#和 Visual Basic 语言的语法中，并采用标准的、易于学习的查询模式。可以对此技术进行扩展以支持几乎任何类型的数据存储。

- **外接程序和扩展性**

.NET Framework 3.5 中的 System.AddIn.dll 程序集向可扩展应用程序的开发人员提供了强大而灵活的支持。它引入了新的结构和模型，可帮助开发人员完成向应用程序添加扩展性的初始工作，并确保开发人员的扩展在宿主应用程序发生更改时仍可继续工作。

- **Windows Presentation Foundation**

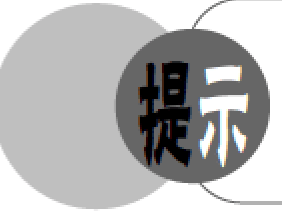
在.NET Framework 3.5 中，Windows Presentation Foundation 包含多个方面的更改和改进，其中包括版本控制、应用程序模型、数据绑定、控件、文档、批注和三维 UI 元素。

- **WCF 和 ASP.NET AJAX 集成**

WCF 与 ASP.NET 中的异步 JavaScript 和 XML（Ajax）功能的集成提供了一个端对端的编程模型，可用于构建可以使用 WCF 服务的 Web 应用程序。在 Ajax 样式的 Web 应用程序中，客户端（例如，Web 应用程序中的浏览器）通过使用异步请求来与服务器交换少量的数据。在 ASP.NET 中集成 Ajax 功能可提供一种生成 WCF Web 服务的简单方法，通过使用浏览器中的客户端 JavaScript 可以访问这些服务。

- **ClickOnce 清单**

新增了一些密码类，用于验证和获取有关 ClickOnce 应用程序的清单签名的信息。



在这里仅列举了.NET Framework 3.5 中的重要新功能和新特性，但不是全部。读者如果需要了解更多，可到网站 http://www.microsoft.com 上查找。

6.1.2 开发环境简介

Visual Studio 是一套完整的开发工具，用于生成 ASP.NET Web 应用程序、XML Web Services、桌面应用程序和移动应用程序。Visual Basic、Visual C#和 Visual C++都使用这一相同的集成开发环境（IDE），这样就能够进行工具共享，并能够轻松地创建混合语言解决方案。

可以使用 Visual Studio 的基于组件的强大开发工具和其他技术，简化企业级解决方案的基于团队的设计、开发和部署。另外，这些语言使用.NET Framework 的功能，它提供了可简化 ASP .Net Web 应用程序和 XML Web Services 开发的关键技术。

Visual Studio 2008 是 Visual Studio 的最新版本，图 6-2 说明了 Visual Studio 2008 与.NET Framework 之间的关系。

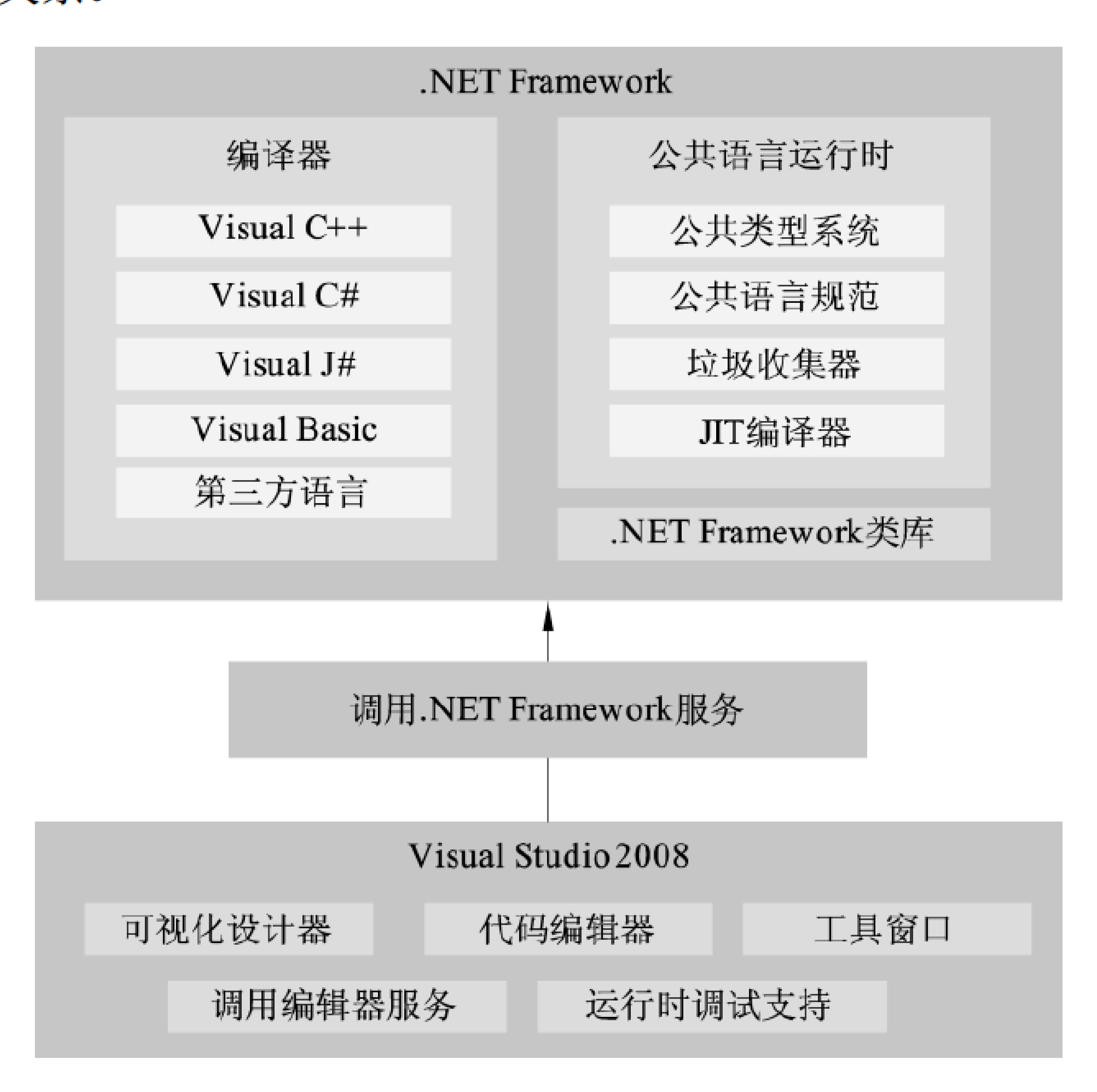


图 6-2 .NET Framework 和 Visual Studio 2008 之间的关系

从图 6-2 中可以看出，Visual Studio 依赖于.NET Framework 提供的服务。这些服务包括 Microsoft 公司或者第三方提供的语言编译器。这些语言编译器是.NET Framework 自身的组成部分，而不属于 Visual Studio。Visual Studio 提供了大量的工具来调用某一种安装的编译器。

Visual Studio 2008 中包含了很多的新特性和新功能，这些全新的功能会大大提高开发人员的工作效率并且减少程序复杂性。这些新功能主要包括：.NET Framework 对重定向的支持；ASP.NET Ajax 和 JavaScript 智能客户端支持；全新的 Web 开发新体验，Web 设计器提供了分

割视图编辑、嵌套母版页以及强大的 CSS 编辑器集成；编程语言方面的改进和 LINQ；浏览.NET Framework 库源码；智能部署 ClickOnce；.NET Framework 3.5 增强功能；集成对 Office（VSTO）和 Sharepoint 2007 开发的支持；在 Windows Server 2008、Windows Vista 和 Microsoft Office 2007 下最好的开发工具集；单元测试功能，所有的 Visual Studio 专业版本都支持单元测试功能；等等。

用户在执行由任何.NET Framework 语言开发的应用程序时，必须安装.NET Framework。不过.NET Framework 会在安装 Visual Studio.NET 程序时自动安装，当然读者也可以从 Microsoft 公司的站点下载免费的.NET Framework。在安装最新的 Visual Studio 2008 时会自动安装所需的.NET Framework 3.5、.NET Framework 3.0 和.NET Framework 2.0 版本。

下面来熟悉一下 Microsoft Visual Studio 2008 的开发环境，首先选择【开始】|【程序】|Microsoft Visual Studio 2008|Microsoft Visual Studio 2008 命令启动程序，第一个出现的是 Microsoft Visual Studio 2008 的启动画面，如图 6-3 所示。

接下来是初始化环境设置，在这里选择默认启动时使用的环境，例如 Windows 应用程序开发人员可以选择 C#或者 VB 作为默认环境，当然环境也可以在启动以后进行修改。这里选择了使用 C#进行开发的设置，如图 6-4 所示。

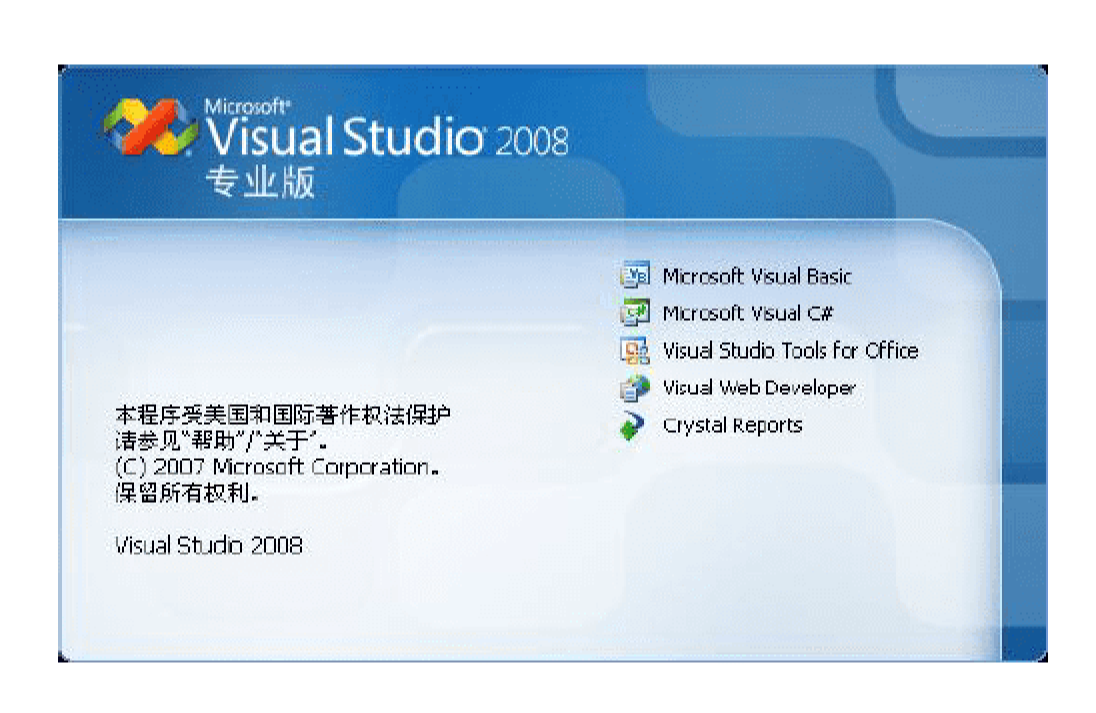


图 6-3　启动画面

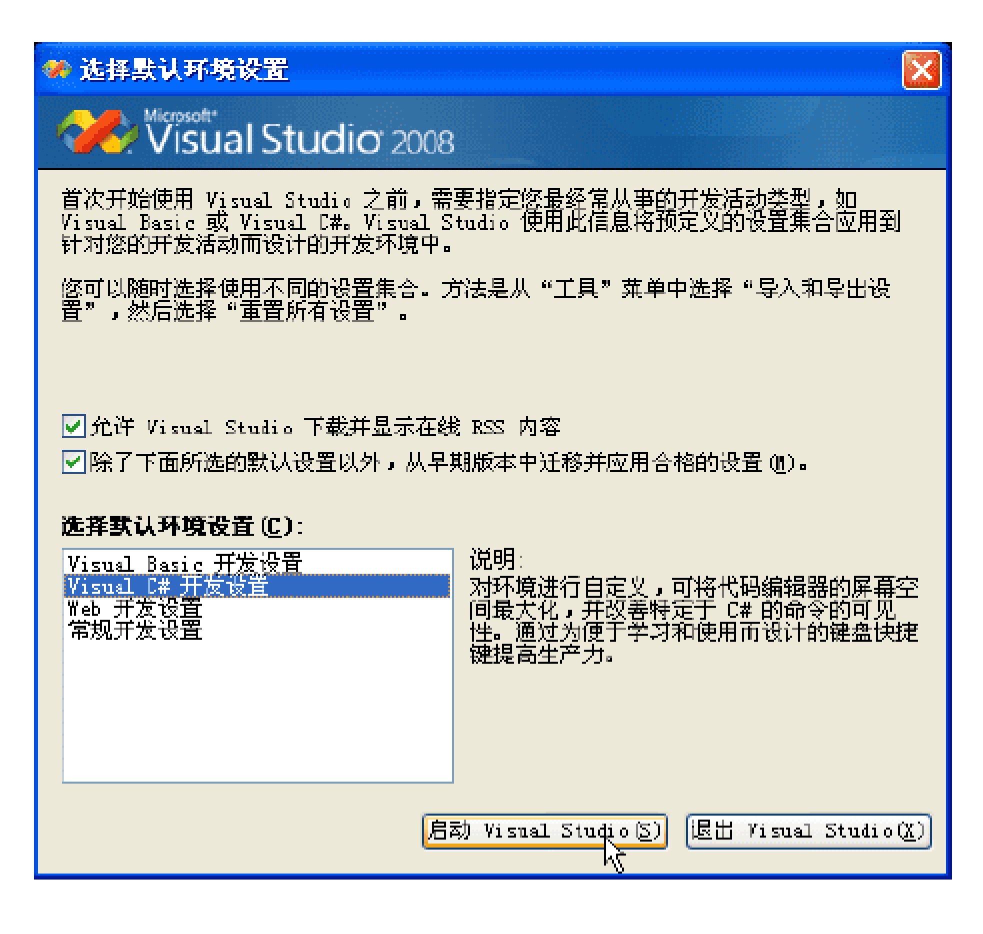


图 6-4　选择默认环境设置

单击【启动 Visual Studio】按钮开始载入程序并执行配置环境操作，待完成后会出现 Microsoft Visual Studio 2008 的主界面。现在，在主界面中选择【帮助】|【关于 Microsoft Visual Studio】命令来查看它的版本信息，此时将弹出一个对话框，如图 6-5 所示。

单击【确定】按钮返回 Visual Studio 2008 的【起始页】窗口，在这里为方便、快速而容易地开始使用 Visual Studio 2008 提供了一种简洁方式，可以轻松打开常用项目、创建新项目、找到联机资源，以及管理 Visual Studio 2008 的配置文件等。

使用 Visual Studio 2008 创建最多的是 Windows 和 Web 应用程序。创建方法为：打开 Visual Studio 2008 的【起始页】窗口后，选择【文件】|【新建】|【项目】命令打开【新建项目】对话框，在这里将看到许多的【项目类别】和【项目模板】以树状视图组织，如图 6-6 所示。

这里要注意，Visual Studio 2008 与之前版本的重大区别就是，对多个.NET Framework 版本的支持，即在图 6-6 所示的对话框中可以选择要创建的应用程序使用的.NET Framework

版本。

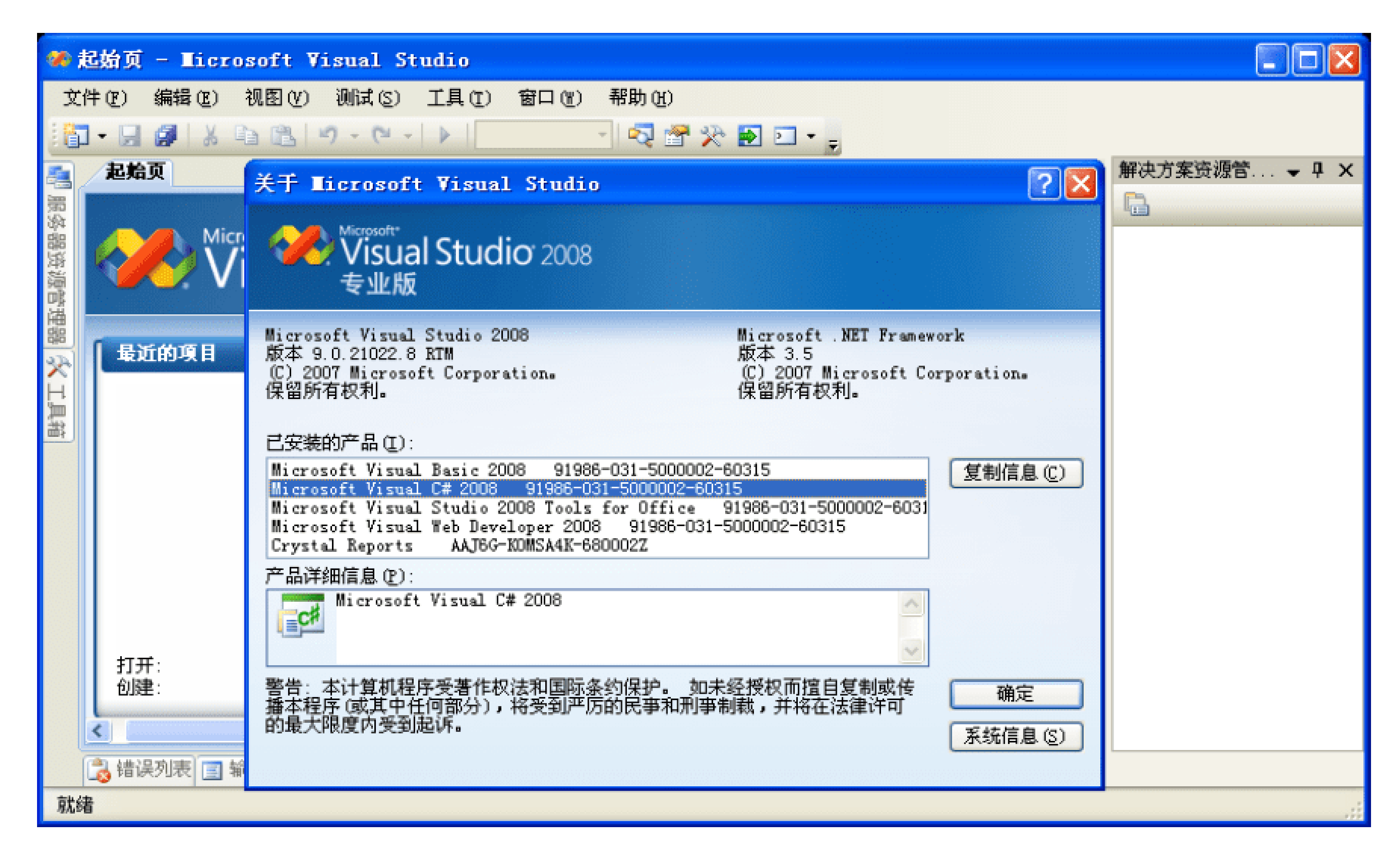


图 6-5 查看版本信息

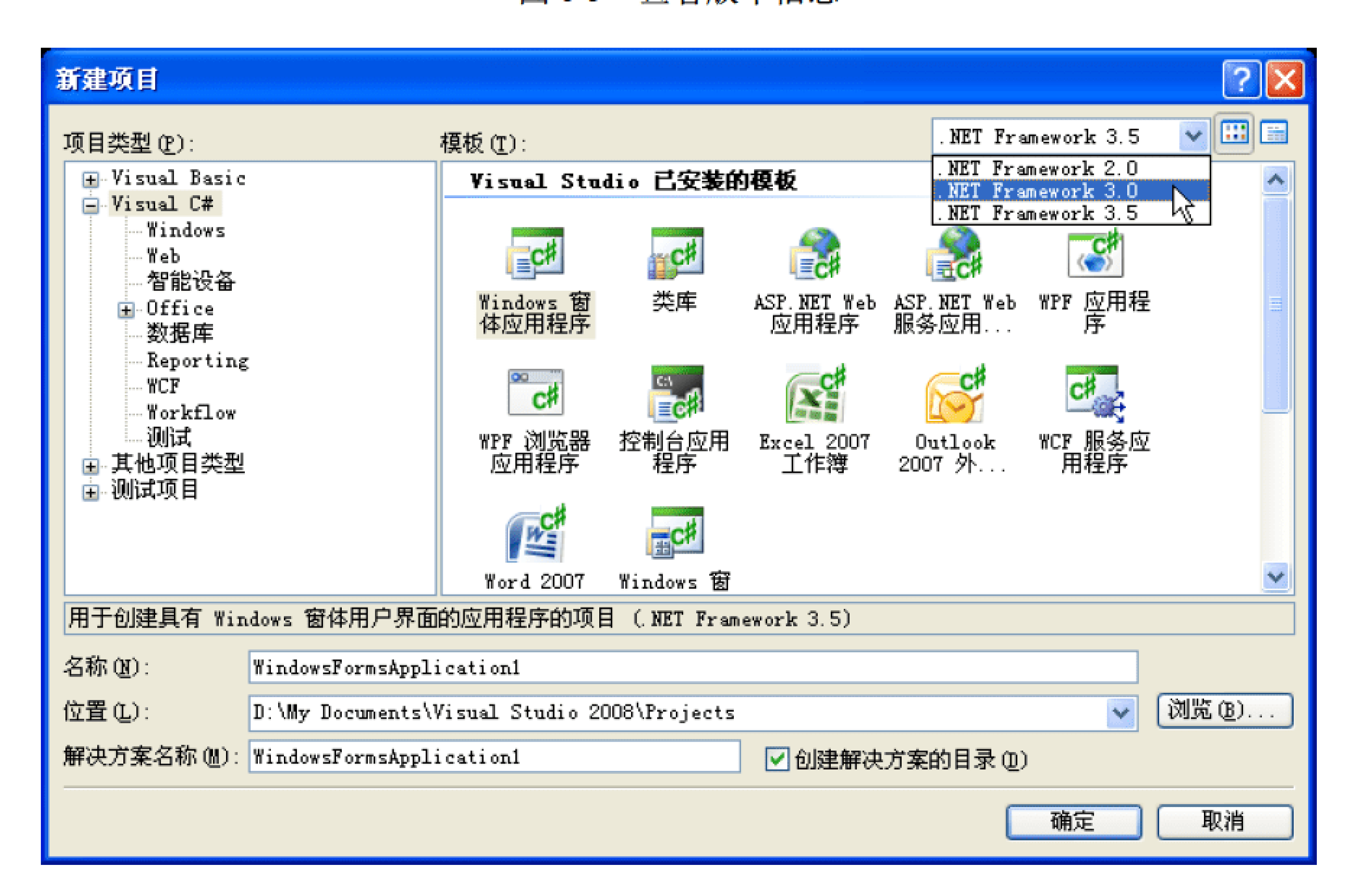


图 6-6 【新建项目】对话框

要创建 Web 站点（包括 ASP.NET 网站、Web 服务和 Crystal Report 等）可以选择【文件】|【新建】|【网站】命令打开【新建网站】对话框，如图 6-7 所示。在这里同样可以选择 Web 网站使用的.NET Framework 版本。

在 Visual Studio 2008 中还提供了一个功能改进的 HTML 和 ASP.NET 网页设计器。此外，这个所见即所得（WYSIYG）设计器还提供了如下特性。

- 分割视图（Split View）的支持（同时将 HTML 源码和所见即所得设计模式打开的能力）。

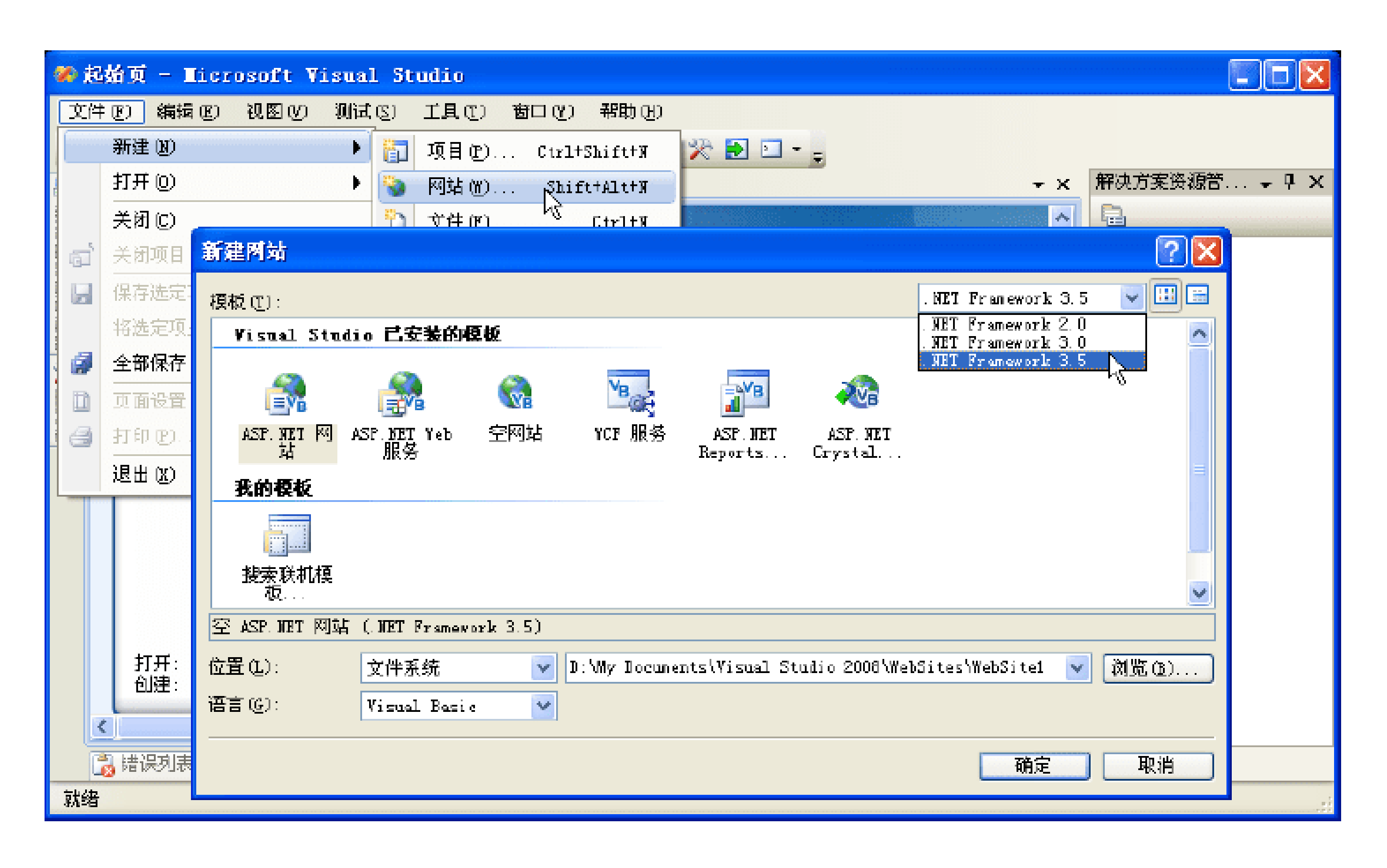


图 6-7 【新建网站】对话框

- 无比丰富的 CSS 支持（CSS 属性窗口、CSS 继承图示器，CSS 预览，以及 CSS 管理器）。
- 极大改进的视图转换性能（从源码模式转换成 HTML 设计模式将会瞬时完成）。
- 对源码视图内控件设计器的支持（属性构造器、事件接通（Wire-up）以及向导将会在源码视图内正常工作）。
- 更丰富的标尺（Ruler）和布局支持（值将被自动储存在外部的 CSS 文件中）。
- 对内嵌母版页（Nested Master Pages）设计器的支持。

图 6-8 所示为是一个在分割视图编辑模式下的 ASP.NET 网页的截图，在这里允许开发人员同时在源码模式和设计模式里操作。另外，Visual Studio 2008 也对 CSS 的支持进行了增强，在新的【管理样式】属性窗口中允许开发人员在样式表内轻易地创建、管理和重构 CSS 规则，就像是使用专业的网页开发和编辑工具一样。

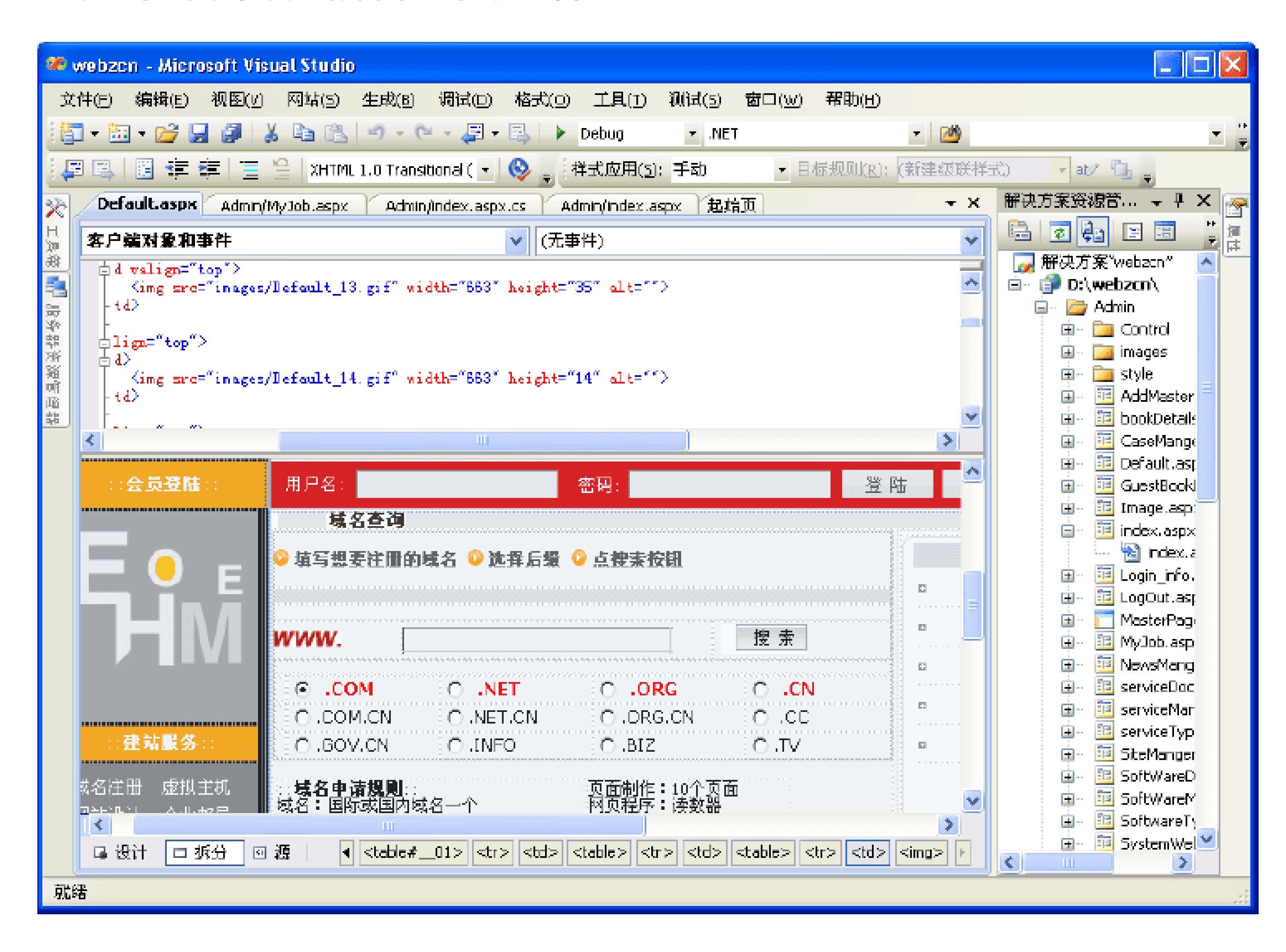


图 6-8　分割视图

Visual Studio 2008 对 ASP.NET 的改变主要是包括了 ASP.NET Ajax 1.0 内置的所有功能，而且还提供了对 Ajax 和 JavaScript 工具的支持，包括 JavaScript 脚本智能提示和更加丰富的调试支持等。如图 6-9 所示，Visual Studio 2008 对标准的行内 JavaScript 脚本自动动态产生的提示信息。

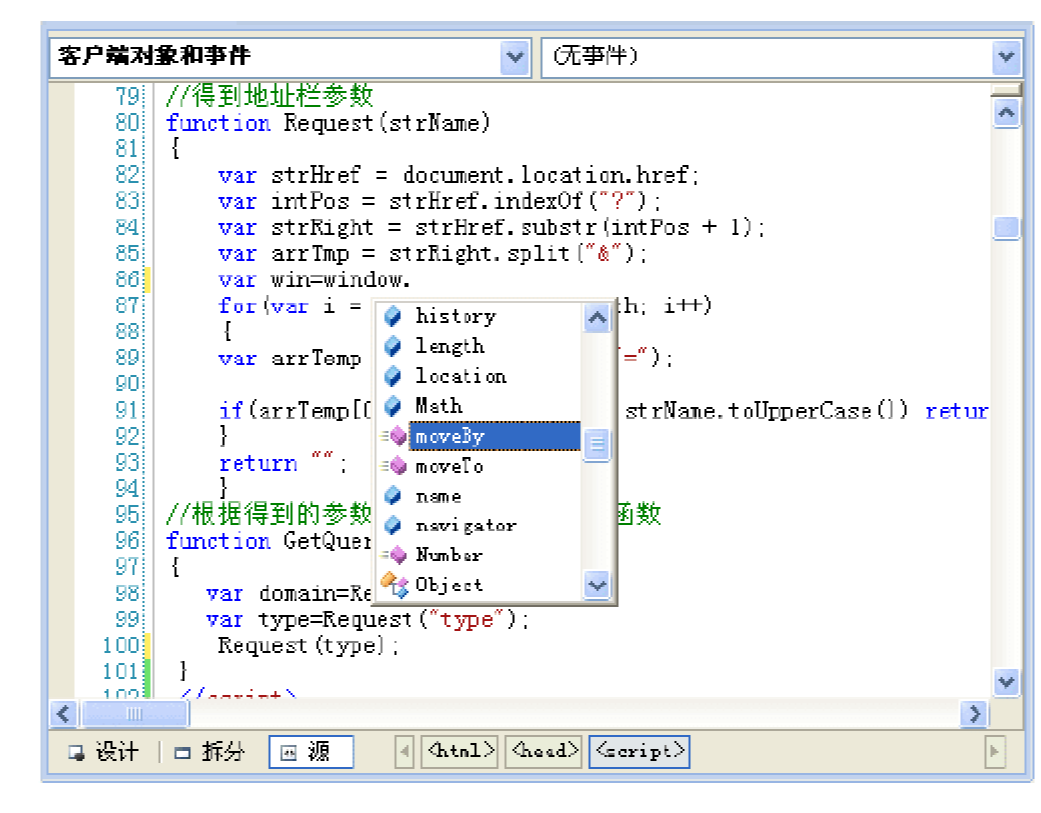


图 6-9　增强的 JavaScript 智能提示

Visual Studio 2008 还包含了许多工具，其中一些是对旧功能的增强，一些是新增功能。总之，这些工具都是为了一个共同的目标：让开发人员在设计时能够更轻松地正确编写代码。

6.2　C# 3.5 语法概述

C#是一种简洁、类型安全的面向对象的语言，开发人员可以使用它来构建在.NET Framework 上运行的各种安全、可靠的应用程序。使用 C#，可以创建传统的 Windows 客户端应用程序、XML Web Services、分布式组件、客户端-服务器应用程序、数据库应用程序以及很多其他类型的程序。Microsoft Visual C# 2008 提供高级代码编辑器、方便的用户界面设计器、集成调试器和许多其他工具，以在 C#语言 3.5 版本和.NET Framework 的基础上加快应用程序的开发。

6.2.1　控制语句

控制语句用于控制程序的流程，以实现程序的各种结构，由特定的语句定义符组成。C#有 9 种控制语句，可以分成 3 类，分别是：条件语句、循环语句和跳转语句。下面详细介绍这几种控制语句。

1. 条件语句

条件语句又称为选择语句，判断一个表达式结果的真假（是否满足条件）。根据结果判断执行哪个语句块。选择语句分为 if 语句和 switch 语句两种。

□ **if 语句**

在 C#语法中，选择语句的语法如下：

```
if (表达式)
{
    //表达式的结果为真时的运行程序段
}else
{
    //表达式的结果为假时的运行程序段
}
```

下面来看一个完整的例子：

```
if(x>y)
{
    Console.WriteLine("x 的值大于 y 的值");
}
else
{
    Console.WriteLine("x 的值小于 y 的值");
}
```

如果 x 的值大于 y 的值，那么结果将会输出：x 的值大于 y 的值。反之则会输出 x 的值小于 y 的值。如果只有一个子语句，可以不使用大括号来括起子语句，如果有多个子语句时，需要用大括号把这些语句组合成一个语句块。在 if 语句中，可以使用 else if 子句判断多条语句。例如：

```
if(表达式 1)
{
    //表达式 1 为真时执行的程序段
}else if (表达式 2)
{
   //表达式 2 为真时执行的程序段
}else if (表达式 N)
{
   //表达式 N 为真时执行的程序段
}else
{
   //表达式 1 为假时执行的程序段
}
```

理论上，if 语句可以合并无限个，但是为了程序更容易理解，建议不要合并的太多，并且每个 if 语句用大括号括起来。

□ **switch 语句**

在 if 语句中，可以看到多个条件判断时的不足，尤其是当需要判断的条件越来越多时，书写非常麻烦。于是 C#提供了用于更多条件判断的语句——switch 语句。switch 语句是一个控制语句，通过控制传递给其体内的一个 case 语句来处理多个选择。其语法如下：

```
switch (caseSwitch)
{
    case 1:
    Console.WriteLine("Case 1");
    break;
    case 2:
    Console.WriteLine("Case 2");
    break;
    default:
    Console.WriteLine("Default case");
    break;
}
```

下面是 switch 多条件判断语句使用的例子，代码内容如代码 6.1 所示。

代码 6.1　使用 switch 语句

```
static void Main(string[] args)
{
    Console.WriteLine("一年中你最喜欢那个季节？");
    string Season = Console.ReadLine();  //从键盘读取用户输入内容
    Console.Write("你喜欢{0}  ", Season);
    switch (Season) //switch 分支
    {
        case "春天":
            Console.WriteLine("这样的天气最适合郊游，相信你肯定喜欢！");
            break;
        case "夏天":
            Console.WriteLine("火辣的天气，去游泳馆会是一个不错的选择！");
            break;
        case "秋天":
            Console.WriteLine("秋高气爽，带上你的家人去旅行吧！");
            break;
        case "冬天":
            Console.WriteLine("白雪茫茫的，去滑雪吧，绝对刺激！");
            break;
        default:  //default 语句
            Console.WriteLine("请确认你输入的季节");
            break;
    }
    Console.Read();
}
```

本例输出的结果如图 6-10 所示。

2. 循环语句

循环语句也称为迭代语句，让程序重复执行某个语句块，直到某个特定的条件表达式结

果为假时，结束执行语句块。C#提供了以下几种可以让程序循环执行的语句。

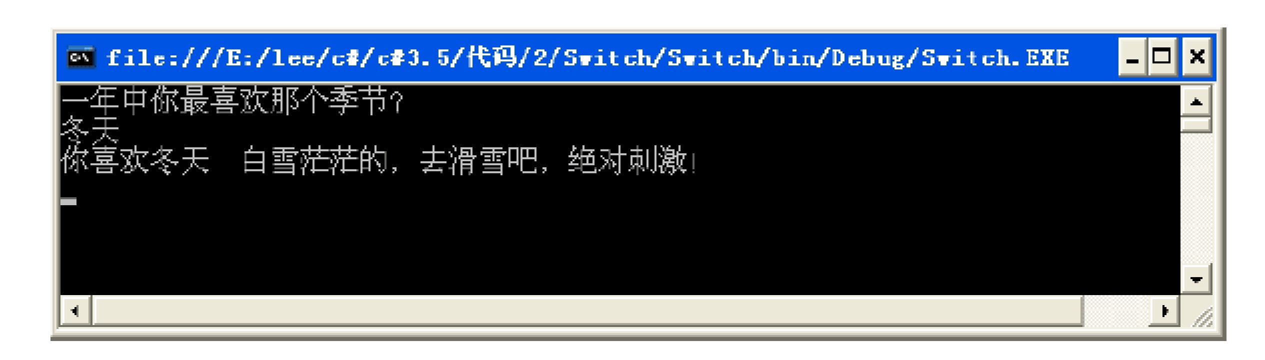


图 6-10　switch 语句执行结果

□　**for 循环**

for 语句是一种在程序执行前就要首先判断条件表达式是否为真的循环语句。如果循环条件为假，那么循环语句根本就不会去执行。for 循环适合用于一个语句或语句块重复执行预定的次数的情况。for 语句通常使用在知道要循环次数的循环中。语法如下：

```
for(初始值表达式; 循环条件表达式; 循环后的操作表达式)
{
    执行语句块
}
```

下面使用 for 循环创建一个“百钱买百鸡”的例子，要求使用 100 元钱购买 100 只鸡，其中公鸡 5 元一只、母鸡 3 元一只、小鸡 1 元 3 只，并且要求这 3 种鸡都必须有。代码内容如代码 6.2 所示。

代码 6.2　使用 for 循环语句

```
class Program
    {
        static void Main(string[] args)
        {
            float a, b, c;
            for (a = 1; a <= 20; a++)
            {
                for (b = 1; b <= 33; b++)
                {
                    c = 100 - a - b;
                    if (5 * a + 3 * b + c / 3 == 100)
                    {
           Console.WriteLine("100 元钱可以买{0}只公鸡, {1}只母鸡, {2}只小鸡",a,
           b, c);
                    }
                }
            }
            Console.Read();
        }
    }
```

程序里声明了 3 个 float 类型变量 a、b、c，嵌套使用 for 语句，循环计算 3 个数的每一种组合，最终使用 if 语句得出正确的结果。本例输出的结果如图 6-11 所示。

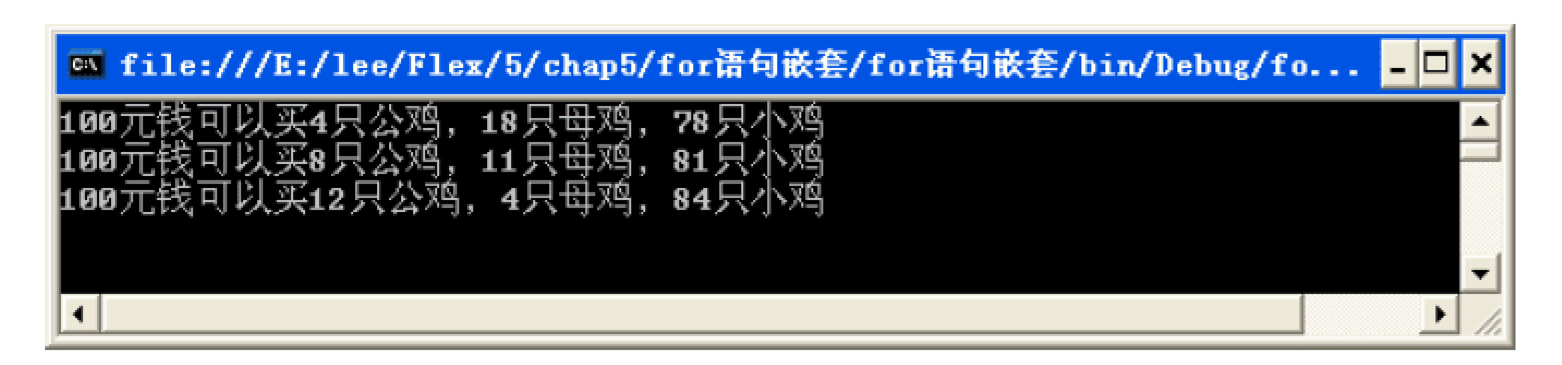


图 6-11　for 循环执行结果

□ **while 循环**

while 循环通常用于下述情况，在循环开始之前不知道重复执行一个语句或语句块的次数。与 for 循环一样，while 循环也是一个预测试循环。但 while 循环只有一个表达式，语法如下：

```
while(表达式)
{
    //执行的语句块
}
```

下面以例子说明 while 语句，代码内容如代码 6.3 所示。

代码 6.3　使用 While 循环

```
static void Main(string[] args)
{
    int i = 1;
    while (i>0)
    {
        i++;
        if (i == 2)
        {
            System.Console.WriteLine(i);
            break;
        }
    }
}
```

在 while 循环中，while 后面是个布尔表达式。所有的控制语句都使用了布尔表达式，这就意味着表达式必须计算出 true 或 false 值。在本例中，i 的初始值是 1，当执行 while 语句是，该布尔表达式返回 true 值。执行结果如下：

```
程序的输出结果是 2。
```

□ **do…while 循环**

do…while 循环是 while 循环测试的扩展。do…while 语句与 while 语句唯一的区别在于，不管表达式的结果为真还是为假，循环语句至少执行一次。因此 do…while 循环适合于至少执行一次循环体的情况，其语法如下：

```
do
{
    //执行的语句块
}
while(表达式)
```


下面以例子说明 do…while 语句的使用，代码内容如代码 6.4 所示。

代码 6.4　do…While 循环

```
static void Main(string[] args)
{
    int i = 0;
    do
    {
        Console.WriteLine(i);
        i++;
    } while (i > 10);
}
```

从代码 6.4 中看到，i 的初始值为 0，那么布尔返回值应该为 false，但还是会输出结果。也就是说，do…while 语句先不管表达式的结果是真是假，都会在执行一次后才会去判断表达式的结果。

- **foreach 循环**

foreach 循环可以迭代出集合中的每一项，但并不能修改集合中的任何一项。foreach 的语法如下：

```
foreach(类型 变量名 in 集合)
{
执行语句;
}
```

类型为集合元素的类型，变量名表示集合中的每一个元素。每次执行一次循环语句，循环变量就读取集合中的一个元素。下面以例子说明 foreach 语句的使用，代码内容如代码 6.5 所示。

代码 6.5　使用 foreach 语句遍历数组

```
static void Main(string[] args)
{
    string[] colors = { "Red", "Blue", "Green", "White", "Yellow" };
    Console.WriteLine("遍历数组的所有颜色: ");
    foreach (string c in colors)
    {
        Console.WriteLine("{0}", c);
    }
    Console.ReadLine();
}
```

其中，foreach 语句循环的参数是由两个元素组成的表达式，用关键字 in 分隔，关键字右边为要访问的元素集合的名称，左边为临时变量。本例的输出结果如图 6-12 所示。

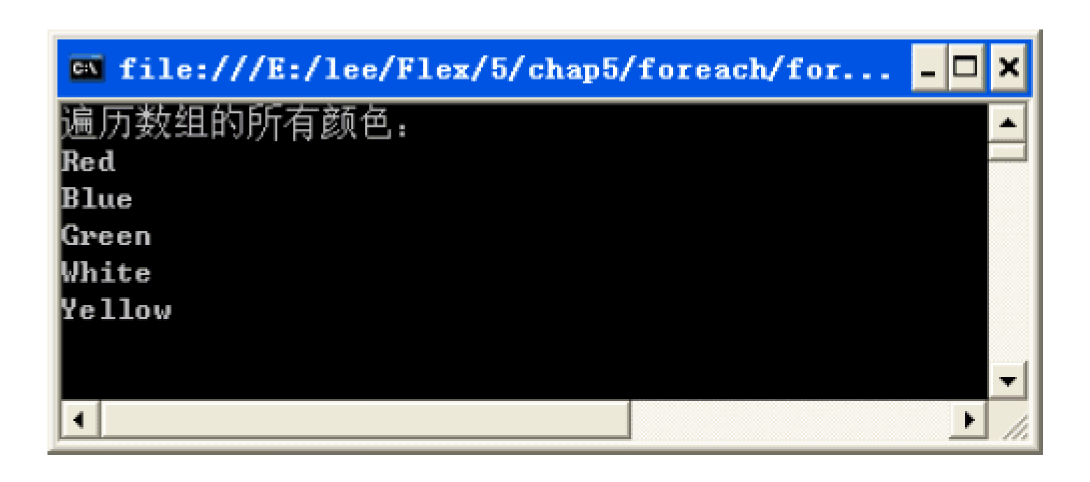
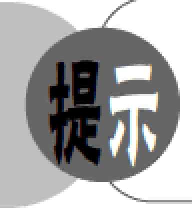


图 6-12　foreach 循环执行结果



临时变量的类型要和元素集合类型兼容。

3. 跳转语句

很多时候，程序需要从一个语句块转到另一个语句块，因此 C#提供了许多可以立即跳转到程序另一行代码执行的语句，这些跳转语句包括：goto 语句、break 语句和 continue 语句。

- **goto 语句**

goto 语句也称为无条件转移语句。goto 语句可以使程序直接跳到程序中使用标签指定的另一行，但也有限制，如不能跳出类的范围，不能跳到像 for 循环那样的语句块中，也不能退出 try…catch 块后的 finally 块（后面会详细地讲解如何用 try…catch…finally 块处理异常）。其语法如下所示。

```
goto 语句标号
```

goto 语句多用于 switch 语句中，实现由某个 switch 的 case 标签或 default 标签跳转到另一个 case 标签或 default 标签。有时也可以用在嵌套循环中，使程序跳出多层循环。下面以例子说明 goto 语句，代码如下所示。

```
static void Main(string[] args)
{
    int myInteger = 5;
    goto myLabel;
    myInteger += 10;
    myLabel:
    Console.WriteLine("myInteger={0}", myInteger);
}
```

上面示例中定义一个 myLabel 标签，当程序执行遇到 goto 语句时将跳转到 myLabel 语句行上，继续执行下面的语句。本例输出结果如下所示。

```
myInteger=5;
```

□ **break 语句**

break 语句使用在循环语句中或条件语句中，用于终止一条循环语句，导致控制流程跳转到循环语句的下一条语句。如在前面讲 switch 条件判断语句时已经多次使用了 break 语句，其作用就是执行完 case 语句块后，将程序的控制权交给 switch 语句后面的第一条语句。下面以例子说明 break 语句的使用，代码内容如代码 6.6 所示。

代码 6.6　使用 break 语句

```
static void Main(string[] args)
{
    Console.WriteLine("输出结果为: ");
    for (int i = 0; i <= 10; i++)
    {
        if (i > 5)
        {
            break;
        }
        Console.Write(i);
    }
}
```

本例的输出结果如下：

```
0 1 2 3 4 5
```

上述例子中，当 if 条件为 false 时，执行 break 语句结束循环。

□ **continue 语句**

continue 语句有些类似于 break 语句，但是只能出现在循环体中。与 break 语句的区别在于：continue 并不是中断循环语句，而是中止当前迭代的循环，进入下一次的迭代。简单地讲，continue 是忽略循环语句的当次循环。下面以例子说明 continue 语句的使用，代码内容如代码 6.7 所示。

代码 6.7　使用 continue 语句

```
static void Main(string[] args)
{
    Console.WriteLine("输出结果为: ");
    for (int i = 1; i <= 8; i++)
    {
        if (i <5)
        {
            continue;
        }
        Console.Write(i);
    }
}
```

本例的输出结果如下：

```
5 6 7 8
```

上述例子中，continue 语句并不会像 break 语句那样干涉循环体重复的次数。当条件不满足时，不执行输出语句直到满足条件时才执行输出语句。

□ **return 语句**

return 语句用于终止所执行的方法，并将控制返回给调用方法。还可以返回一个可选值，该值可以是任何类型变量、结果集等。如果方法为 void 类型，则可以省略 return 语句。下面以例子说明 return 语句的使用，代码内容如代码 6.8 所示。

代码 6.8　使用 return 语句

```
class Program
{
    static int add(int x,int y)
    {
      return x+y;
    }
    static void Main(string[] args)
    {
      int sum=add(11,32);
      Console.WriteLine("两个数相加的结果是：{0}",sum);
    }
}
```

屏幕输出的结果如下：

```
两数相加的结果是：43
```

上述例子中定义一个静态的方法计算两数相加。在主函数中调用 add()这个方法，并传递两个参数，然后计算出结果。

6.2.2　面向对象实现

面向对象方法（Object Oriented Method）是一种把面向对象的思想应用于软件开发过程中，指导开发活动的系统方法，简称 OO（Object Oriented）方法。OO 方法是建立在“对象”概念基础上的方法学。对象是由数据和容许的操作组成的封装体，与客观实体有直接对应关系。一个对象类定义了具有相似性质的一组对象。而继承是对具有层次关系的类的属性和操作进行共享的一种方式。所谓面向对象就是基于对象概念，以对象为中心，以类和继承为构造机制，来认识、理解、刻画客观世界和设计、构建相应的软件系统。

对象（Object）即指现实世界中各种各样的实体，可以指具体的事物也可以指抽象的事物。如：整数 1、2、3、人、树、规则、法律、书本等。每个对象皆有自己的内部状态和运动规律，比如狗具有体重、种类等内部状态，具有叫、跑等运动规律。在面向对象概念中把对象的内部状态称为属性，运动规律称为方法或事件。

1. 类

类是 C#中功能最为强大的数据类型。像结构一样，类也定义了数据类型的数据和行为。然后，程序员可以创建此类的实例对象。与结构不同，类支持继承，而继承是面向对象编程的基础部分。C#中的类是一种数据结构，一般成员包括：数据成员、函数成员、嵌套类型等。

声明语法如下：

```
[属性 类修饰] class 类名称 : [基类规范]      //中括号中的元素为可选元素
```

例如：

```
public partial class _Default : System.Web.UI.Page
```

类修饰符决定了类在程序运行中被处理的方式。创建类时，可以接受默认的修饰符，也可以根据需要指定一个或多个修饰符。修饰符可以是一个访问修饰符（public，protected，private），加上一个或多个类型修饰符（abstract、static、final、strictfp）。C#类中提供的类修饰符详见表 6-1。

表 6-1　类修饰符

关键字	分类	说明
new	访问修饰符	适用于嵌套类，被修饰的类会把继承下来的同名成员隐藏
public		存取不受限制
private		只有包含该成员的类可以存取
internal		只有当前工程可以存取
protected		只有包含该成员的类以及继承的类可以存取
abstract		可以被指示一个类只能作为其他类的基类
sealed		指示一个类不能被继承
abstract	成员修饰符	指示该方法或属性没有实现
const		指定域或局部变量的值不能被改动
event		声明一个事件
extern		指示方法在外部实现
override		对由基类继承成员的新实现
readonly		指示一个域只能在声明时以及相同类的内部被赋值
static		指示一个成员属于类型本身，而不是属于特定的对象
virtual		指示一个方法或存取器的实现可以在继承类中被覆盖

下面以例子说明。

```
public string name="Jim";
public string Name
{
    get { return name; }
    set { name = value; }
}
```

在本例中，Name 作为属性声明，Name 属性使用私有字段来跟踪实际值。属性的数据的真实位置经常称为属性的“后备存储”。属性使用作为后备存储的私有字段很常见。将字段标

记为私有可确保该字段只能通过调用方法来更改。

方法在类或结构中声明。声明时须要指定访问级别、返回值、方法名称以及任何方法参数。方法参数放在括号中，并用逗号隔开。空括号表示方法不需要参数。

下面的类包含 3 个方法：

```
class Program
{
    public void test() {}
    public void test(int gallons) {}
    public string test(bool flag){}
}
```

C#中的方法其实是一个功能块，语法如下：

```
[属性 类修饰] 返回值类型 函数名称(传递参数列表){功能程序块}
```

接下来实现 Program 类中的 test(bool flag)方法，如下所示：

```
public string test(bool flag)
{
if(flag)
return "hello world!";
else
return "my name is Jim!";
}
```

2. 抽象

声明一个抽象方法使用 abstract 关键字。一个类中可以包含一个或多个抽象方法。抽象类中可以存在非抽象的方法但不能被直接实例化。实现抽象类用“:”（冒号），实现抽象方法用 override 关键字。抽象类可以被抽象类所继承，结果仍是抽象类。抽象方法被实现后，不能更改修饰符。下面以例子说明抽象的使用，代码内容如代码 6.9 所示。

代码 6.9　抽象类

```
class Program
{
    static void Main(string[] args)
    {
        new Name().Jim();
    }
}
public abstract class Person
{
    public abstract void Jim();

}
```

```
public class Name : Person
{
    public override void Jim()
    {
        Console.WriteLine("my name is Jim");
        Console.ReadLine();
    }
}
```

输出结果如下：

```
my name is Jim
```

3. 重载

一个类型上可以存在多个同名的方法。当出现这种方法时，必须按照某种明显的方式来区分，将这种特性称为重载。重载类的同名方法在给其传递不同的参数时可以有不同的运动规律。在对象间相互作用时，即使接收消息对象采用相同的接收办法，但消息内容的详细程度不同，接收消息对象内部的运动规律也可能不同。

函数重载是指在同一作用域内的若干参数特征不同的函数可以使用相同的函数名字，运算符重载是指同一个运算符可以施加于不同类型的操作数上面。重载对于提高系统的灵活性和可读性起到了很好的作用。

方法重载，指在类中创建了多个方法，具有相同的方法名，但有不同的参数、不同的返回类型或不同的方法体。代码 6.10 演示了方法重载的使用。

代码 6.10　方法重载

```
public class Max
{
    public int sum(int x, int y)
    {
        return x * y;
    }
    public double sum(int x)
    {
        return 3.14159 * x * x;
    }
    public static void Main(String[] args)
    {
        Max mymax = new Max();
        Console.WriteLine("圆的面积为" +mymax.sum(3));
        Console.WriteLine("长方形的面积为" +mymax.sum(4,7));

    }
}
```

本例的输出结果如下：

```
圆的面积为 28.27431
长方形的面积为 28
```

4．面向对象特征

面向对象由 3 种基本特征，即封装、继承和多态。下面将详细介绍。

- **封装**

封装是面向对象的特征之一，是对象和类概念的主要特征。封装也就是把客观的事物封装为抽象的类。简洁的讲，封装是一种信息隐藏技术。封装使数据和加工该数据的方法（函数）封装为一个整体，以成为独立性很强的模块，使得用户只能见到对象的外特性（对象能接受哪些消息，具有哪些处理能力），而对象的内特性（保存内部状态的私有数据和实现加工能力的算法）对用户隐蔽。封装的目的在于把对象的设计者和对象的使用分开，使用者不必知晓行为实现的细节，只须用设计者提供的消息来访问该对象。

- **继承**

继承就是在类之间建立一种相交关系，使得新定义的派生类可以继承已有基类的特征和能力，而且可以加入新的特性或者是修改已有的特性建立起类的新层次。一个类从另一个类派生出来时，派生类从基类那里继承特性。派生类亦可以作为其他类的基类。从一个基类派生出来的多层类，形成了类的层次结构。继承分为单继承（一个派生类只有一个基类）和多重继承（一个派生类有多个基类）。类的对象各自封闭，如果没有继承性机制，则类对象中的数据、方法就会出现大量重复。继承不仅支持系统的可重用性，而且还增强系统的可扩充性。

继承是面向对象程序设计的主要特征之一，可以重写代码，从而节省程序设计的时间。类的继承示例如代码 6.11 所示。

代码 6.11　类的继承

```
using System;
public class ParentClass
{
    public ParentClass()
    { Console.WriteLine("这是父类构造函数输出。"); }
    public void print()
    { Console.WriteLine("I'm a Parent Class。"); }
}
public class ChildClass : ParentClass
{
    public ChildClass()
    { Console.WriteLine("这是子类构造函数输出。"); }
    public static void Main()
    {
        ChildClass child = new ChildClass();
        child.print();
```

```
        }
    }
```

输出结果如下所示。

```
这是父类构造函数输出
这是子类构造函数输出。
I'm a Parent Class。
```

□ 多态

面向对象程序设计的另外一个重要概念是多态性。通过继承，一个类可以用作多种类型：可以用作自己的类型、任何基类型，或者在实现接口时用作任何接口类型。这称为多态性。C#中的每种类型都具有多态性。类型可用作自己的类型或用作 Object 实例，因为任何类型都自动将 Object 当作基类。

多态性不仅对派生类很重要，对基类也很重要。在任何情况下，使用基类实际上可能是在使用已经强制转化成基类类型的派生类对象，基类的设计者预测到该基类中可能会在派生类时发生更改的方面。

当派生类从基类继承时，会获得所有基类的方法、属性等，若要改变基类的行为，有两种方法：一是使用新的派生成员替代基成员，二是重写基类中的虚拟成员。

在使用新的派生成员替换基成员时需要使用关键字 New。如果基类定义了一个方法或字段，则可用 New 关键字创建该方法或字段的新定义，New 关键字要放在返回值类型之前使用。

6.2.3　结构

利用以往介绍过的简单类型，进行一些常用的数据运算、文字处理似乎已经足够。但是经常碰到一些更为复杂的数据类型。比如，电话簿的记录中可以包含他人的姓名、电话和地址。如果按照简单类型来管理，每一条记录都要存放到 3 个不同的变量当中，这样工作量很大，也不够直观。有没有更好的办法呢？在实际生活中，一组相关的信息经常被放在一起。一系列相关的变量被组织成为一个单一实体的过程，称之为生成结构的过程。这个单一实体的类型就叫做结构类型，每一个变量称为结构的成员。结构类型的变量采用 struct 来进行声明，语法格式如下：

```
struct 类型名称
{
//主体
}
```

根据上面的分析可以定义电话簿记录结构，其定义方式如下：

```
struct PhoneBook
{
public string name;
public string phone;
public string address;
```

```
}
PhoneBook PhoneNumber1;
```

PhoneNumber1 就是一个 PhoneBook 结构类型的变量。上面声明中的 public 表示对结构类型的成员的访问权限，对结构成员的访问通过结构变量名加上访问符“.”，再跟成员的名称，格式如下：

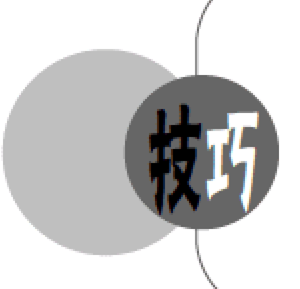
```
PhoneNumber1.name
```

结构类型包含的成员类型没有限制，可以相同，也可以不同。例如可以在电话簿的记录中再加上年龄这个成员，如下：

```
struct PhoneBook
{
public string name;
public int age;
public string phone;
public string address;
}
```

甚至可以把结构类型作为另一个结构的成员的类型，这也没有任何问题。

```
struct PhoneBook
{
public string name;
public int age;
public string phone;
public struct address
{
public string city;
public string street;
public int no;
}
}
```



结构和类的区别：不能为结构声明一个自己的默认构造函数，因为编译器将始终生成一个默认构造函数，总是将字段设置为 0、false、null。所有的构造函数必须显式地初始化所有字段。类，可以在声明的同时初始化实例字段，但在 struct 中不能这么做。

6.2.4 枚举

Enum 类型（枚举类型）是一组命名常量的集合，或者说是用户定义的整数类型的集合。在声明一个枚举时，要指定该枚举可以包含的一组可接受实例值。同时可以创建一系列容易

记忆的名称，让程序更容易理解。每一种枚举都有一种类型，除 Char 之外的所有整型都可以作为枚举类型的基本类型。那么如何声明一个枚举呢？方法如下：

```
[Modifiers]enum wskger{enum-list}
```

上述枚举的声明方法中各个参数的含义：Modifiers 是可以选用的修饰符（包括 New 在内的 4 种修饰符），enum-list 是枚举的成员名称。多个成员之间用逗号隔开。在声明枚举的时候，可以为每个成员赋初值。如果没有赋初值，成员的类型默认为 int 型。默认的第一个成员的值为 0，后面的成员依次加 1。例如：

```
Enum wskger{list1,list2,list3,list4}
```

如果此时要输出枚举的各个成员的值时，结果如下：

```
List1=0
List2=1
List3=2
List4=3
```

代码 6.12 详细地说明枚举的用法。

代码 6.12　使用枚举类型

```
class program
{
    public enum wskger
    {
        list1 = 10,
        list2 = 15,
        list3 = 5,
        list4 = 20
    }
 static void Main()
{
int vall1 = (int)wskger.list1;
int vall2 = (int)wskger.list2;
int vall3 = (int)wskger.list3;
int vall4 = (int)wskger.list4;
Console.WriteLine(vall1);
Console.WriteLine(vall2);
Console.WriteLine(vall3);
Console.WriteLine(vall4);
Console.Read();
 }
}
```

屏幕输出的结果如下：

```
10 15 5 20
```

上述例子没有声明枚举的类型，其枚举的类型为默认的 int 类型。枚举访问时所用的修饰符遵循访问修饰符和类或结构的访问修饰符的规则。声明为命名空间成员的枚举可以有 public 或 internal 访问属性。枚举使用 new 关键字修饰时，则表示它隐藏一个被继承的成员。并且枚举不能派生。

6.2.5 数组和集合

存储相关数据项是大多数软件应用程序的一项基本要求；这可以通过使用数组和集合这两种主要方式来实现。数组是相同类型的对象的集合，数组声明了容纳元素的类型，而集合不声明，这是由于集合以 object 类型来存储元素。一个数组实例具有固定的大小，不能伸缩。集合则可根据需要动态改变大小。数组是一种可读/可写数据结构——没有办法创建一个只读数组。然而可以使用集合提供的 ReadOnly()方法，以只读方式来使用集合，该方法将返回集合的只读版本。

1. 数组

数组是相同类型的对象的集合。数组实际上可以是任意长度，这意味着可以存储数千乃至数百万个对象，但其大小必须在创建数组时就确定下来。数组中的每一项都由一个索引来访问，索引只是一个指示对象在数组中的存储位置或槽的数字。数组既可用于存储引用类型，也可用于存储值类型。

数组根据存储数据的形式可分为一维数组和多维数据，数组是一个经过索引的对象集合。一维数组结构相对比较简单，以线性方式存储了固定数目的项，它仅仅需要一个索引值就可以确定任何一项。

在 C#中，数组声明中的方括号必须紧跟数据类型，而不可以像在 Java 中一样出现在变量名的后面。因此，可以使用下面的语法来声明整数类型的数组。

```
int[] MyArray;
```

而下面的声明在 C#中是无效的。

```
int MyArray[];
```

一旦声明了数组，就可以使用新的关键字来设置它的大小，例如：

```
int[] MyArray;       //声明数组
MyArray=new int[5]; //设置他包含 5 个 int 对象的大小
```

数组声明完成后，就可以访问一维数组中的元素，C#数组的索引也是从零开始的。

```
MyArray[4]           //数组中的最后一个对象
```

初始化数组时可以使用与 Java 相同的语法在创建时对数组元素进行初始化。

```
MyArray=new int[5]{1,2,3,4,5};
```

与 Java 不同，初始化器的数目必须与数组大小完全匹配。也可以利用这一特性在一行中声明和初始化 C#数组。

```
int[] TaxRates={0,20,23,40,50};
```

在上述代码中，创建了一个大小与初始化器的数目相等的数组。

还可以在程序循环中初始化，C#中初始化数组的另一种方法就是使用 foreach 循环。下面的循环将数组中的每个元素都设置为零。

```
int[] MyArray=new int[5];
foreach(int i in MyArray)
{
MyArray[i]=0;
}
```

在介绍完一维数组后接下来讲解多维数组的使用。从概念上来说，具有二维的多维数组类似于一个网格，而具有三维的多维数组类似于一个立方体。C#允许创建规则的多维数组，它可以看作是相同类型的值的矩阵。数组可以具有多个维度。

下面声明创建一个四行两列的二维数组。

```
int[,] array=new int[4,2];
```

下列声明创建一个三维（4、2 和 3）数组。

```
int[,,] array1=new int[4,2,3];
```

可以在声明数组时将其初始化，如下例所示。

```
int[,] array2D=new int[,]{{1,2},{3,4},{5,6},{7,8}};
int[,,] array3D=new int[,,]{{{1,2,3}},{{4,5,6}}};
```

也可以初始化数组但不指定级别。

```
int[,] array4={{1,2},{3,4},{5,6},{7,8}};
```

如果选择声明一个数组变量但不将其初始化，必须使用 new 运算符将一个数组分配给此变量。例如：

```
int[,] array5;
array5=new int[,]{{1,2},{3,4},{5,6},{7,8}};
```

也可以给数组元素赋值，例如：

```
array5[2,1]=25;
```

下面的代码示例将数组变量初始化为默认值（交错数组除外）。

```
int[,] array6=new int[10,10];
```

多维数组的一种变体是交错数组，即由数组组成的数组。交错数组是一维数组，且每个

元素自身是一个数组。作为元素的数组无需均为相同的大小。

声明交错数组的方式如下：

```
int[][] jaggedArray=new int[3][];
```

这样做会创建一个有 3 个数组的数组。这些数组可以按如下方式初始化。

```
jaggedArray[0]=new int[5];
jaggedArray[1]=new int[4];
jaggedArray[2]=new int[2];
```

2. 集合

数组这个数据结构有一定的局限性，一旦创建好数组，就不能对它的大小进行改变。换句话说就是数组一旦创建好就固定了大小，不可能在数组的末尾新添元素。如果必须对现有数组大小进行改变就只能新建一个数组，这意味着用于处理数组的语法比较复杂。为了简化处理数组的语法，这里引入了集合。集合是创建在 C#内部并且可以进行一些处理的类。

集合相对于数组而言，语法已经标准化。这使得利用集合进行处理要比数组简单得多。集合的功能通过执行 System.Collections 命名空间中的接口来实现。

在 C#.NET 中，集合都实现了 ICollection 接口。而 ICollection 接口是继承自 IEnumerable 接口，那么每个内建的集合也因此实现了 IEnumerable 接口。

C#中提供了很多集合，换个角度说，这些集合都在 System.Collections 命名空间中。每个集合都实现了 ICollection 接口。表 6-2 中列出了该接口的成员。表 6-3 中列出了集合的接口。

表 6-2　ICollection 接口的成员

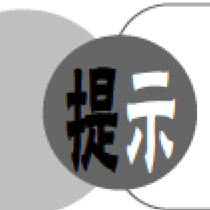
成员	说明
GetEnumerator	该方法返回一个 Enumerator，以此来遍历整个集合。这个方法从 IEnumerable 接口继承得到
Count	该属性能够得到集合中元素的数量
IsSynchronized	该属性表明这个类是否是线程安全的
SyncRoot	可以使用该属性来使对象与集合同步
CopyTo	该方法将集合的元素复制到一个数组中

表 6-3　集合接口

接口	说明
ICollection	为 C#中实现的所有集合提供一个标准接口
IComparer	使集合能够对集合中的项进行排序
IDictionary	表示包含键/值对的集合
IDictionaryEnumerator	使键/值对集合能够对集合中的项进行排序
IEnumerable	使集合能够对集合中的项进行遍历
IList	可以按索引访问集合

下面讨论一下集合的遍历。有一种方法就是用 IEnumerable.GetEnumerator()方法来返回一个枚举数，然后用这个枚举数来遍历整个集合。这建立在所有的集合都实现了 IEnumerable 接

口的基础上。具体的方法是：先实例化枚举数，这时枚举数被放在集合的第一个元素之前，然后调用 MoveNext 方法让枚举数移向集合的下一个元素，当枚举数到达集合的末尾时，就停留在集合最后一个元素的后面。这样就完成了通过枚举数来遍历集合。



枚举数是一个对象，只能用来读取集合的值，不能用来改变集合的内容。表 6-4 中的内容是枚举数的方法以及属性。

表 6-4　枚举数的方法和属性

名称	说明
Current	该属性返回集合中的当前对象
MoveNext	该方法将枚举数移向集合的下一项
Reset	该方法将枚举数移动到初始位置

通过枚举数遍历集合的代码如代码 6.13 所示。

代码 6.13　通过枚举数遍历集合

```
protected void Print(System.Collections.集合类型 list)
{
    IEnumerator enumerator=list.GetEnumerator();
    //有时是 IDictionaryEnumerator enumerator=list. GetEnumerator();
    while(enumerator.MoveNext())
    {
        Console.WriteLine((string)enumerator.Current);
        //相应地，这句代码有时是 Console.WriteLine((string)enumerator.Value);
    }
}
```

从代码 6.13 可以看出：用枚举数来实现集合的遍历很简单。仅需一个实例化枚举数语句加一个循环就可以了（循环条件是调用枚举数的 MoveNext()方法，直到枚举数停留在集合最后一个元素的后面）。

6.3　配置应用程序

Web.config 文件是一个 XML 文本文件，用来储存 ASP.NET Web 应用程序的配置信息，出现在应用程序的每一个目录中。当通过 C#.NET 新建一个 Web 应用程序后，默认情况下会在根目录下自动创建一个默认的 Web.config 文件，包括默认的配置设置，所有的子目录都继承它的配置设置。如果需要修改子目录的配置设置，可以在该子目录下新建一个 Web.config 文件。它可以提供除从父目录继承的配置信息之外的配置信息，也可以重写或者修改父目录中定义的设置。

6.3.1 ASP.NET 配置概述

使用 ASP.NET 配置系统的功能，可以配置整个服务器上的所有 ASP.NET 应用程序、单个 ASP.NET 应用程序、各个页面或者应用程序子目录。还可以配置各种功能，如身份验证模式、页缓存、编译器选项、自定义错误、调试和跟踪选项等。

使用 ASP.NET 配置系统的功能，可以配置整个服务器、ASP.NET 应用程序或者单独的页面。ASP.NET 配置数据存储在全部命名为 Web.config 的 XML 文本文件中，Web.config 文件可以出现在 ASP.NET 应用程序的多个目录中。使用这些文件，可以在将应用程序部署到服务器上之前、期间或者之后方便地编辑配置数据。可以通过使用标准的文本编辑器、ASP.NET MMC 管理单元、网站管理工具或者 ASP.NET 配置 API 来创建和编辑 ASP.NET 配置文件。ASP.NET 配置文件将应用程序配置设置与应用程序代码分开。通过将配置数据与代码分开，可以方便地将设置与应用程序关联，在部署应用程序之后根据需要更改设置，以及扩展配置架构。

在许多应用程序中，需要存储并使用对用户唯一的信息。用户访问站点时，可以使用已存储的信息向用户显示 Web 应用程序的个性化版本。个性化应用程序需要大量的要求：必须使用唯一的用户标识符存储信息，能够在用户再次访问时识别用户，然后根据需要获取用户信息。若要简化应用程序，可以使用 ASP.NET 配置文件功能，该功能可以执行所有上述任务。

每个 Web.config 文件都将配置设置应用于它所在的目录以及它下面的所有子目录。可以选择用子目录中的设置重写或者修改父目录中指定的设置。通过在 location 元素中指定一个路径，可以选择将 Web.config 文件中的配置设置应用于个别文件或者子目录。

运行时，ASP.NET 使用 Web.config 文件按层次结构为传入的每个 URL 请求计算唯一的配置设置集合。这些设置只计算一次，随后将缓存在服务器上。ASP.NET 检测对配置文件进行的任何更改，然后自动将这些更改应用于受影响的应用程序，而且大多数情况下会重新启动应用程序。只要更改层次结构中的配置文件，就会自动计算并再次缓存分层配置设置。除非 processModel 节已更改，否则 IIS 服务器不必重新启动，所做的更改即会生效。

应用程序运行时，ASP.NET 会创建一个 ProfileCommon 类，该类是一个动态生成的类，从 ProfileBase 类继承而来。动态的 ProfileCommon 类包括根据在应用程序配置中指定的配置文件属性定义创建的属性。然后，会将此动态 ProfileCommon 类的实例设置为当前 HttpContext 的 Profile 属性的值，并且可在应用程序的页面中使用。并可以收集要存储的值，并将其赋值给已定义的配置文件属性。例如，应用程序的主页可能包含提示用户输入邮政编码的文本框。用户输入邮政编码时，可以设置 Profile 属性，以存储当前用户的值。

使用 ASP.NET 配置系统所提供的工具来配置应用程序比使用文本编辑器简单，因为这些工具包括错误检测功能。ASP.NET 配置系统提供一个完整的托管接口，使用该接口，可以通过编程方式配置 ASP.NET 应用程序，而不必直接编辑 XML 配置文件。ASP.NET 配置系统有助于防止未经授权的用户访问配置文件。ASP.NET 将 IIS 配置为拒绝任何浏览器访问 Machine.config 或者 Web.config 文件。

使用 Web.config 配置文件具体有如下优点。

- **配置设置的易读性**　所有的配置信息都存储在 XML 文本文件中，可以使用文本编辑器或者 XML 编辑器（如 Visual Studio.NET）来直接编辑配置文件以进行查看和修改。

- **更新的及时性**　ASP.NET 应用程序配置的更新非常及时，无须重启 Web 服务器，就能使配置应用于正在运行的系统，对终端用户完全透明。
- **无须访问本地服务器**　在更新配置系统时，ASP.NET 可以自动探测配置文件的变化，然后创建一个新的应用程序实例。终端用户被重定向到这个新的应用程序，无须访问本地服务器，配置的改变就可投入应用。
- **易于复制**　ASP.NET 配置文件是 XML 格式，因此可以简单地将 IIS（Internet 信息服务）中的 Web 应用程序文件复制到其他合适的位置。
- **保护配置文件**　ASP.NET 通过配置 IIS 阻止对直接的配置文件浏览器访问，从而保护配置文件不受外部访问。
- **可扩展性**　ASP.NET 的配置系统有很强的扩展性。用户可以自定义新的配置参数，并通过编写相应的处理程序来处理。

6.3.2　Web.config 结构

在 ASP.NET 应用程序中，所有的 ASP.NET 配置信息都储存在 Web.config 文件中的 configuration 元素中。此元素中的配置信息分为两个主区域：配置节处理程序声明区域和配置节设置区域。

1. 配置节处理程序声明

配置节处理程序声明区域驻留在 Web.config 文件中的 configSections 元素内。它包含在声明节处理程序的 ASP.NET 配置 section 元素。可以将这些配置节处理程序声明嵌套在 sectionGroup 元素中，以帮助组织配置信息。通常，sectionGroup 元素表示要应用配置设置的命名空间。例如，所有的 ASP.NET 配置节处理程序都在 system.web 节组中进行分组，如下面的代码实例所示。

```
<sectionGroup name="system.web.extensions" type="System.Web.Configuration.
SystemWebExtensionsSectionGroup, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35">
```

配置节设置区域中的每个配置节都有一个节处理程序声明。节处理程序是用来实现 ConfigurationSection 接口的.NET Framework 类。节处理程序声明中包含配置设置节的名称（如 roleService）以及用来处理该节中配置数据的节处理程序类的名称（如 System.Web.Configuration.ScriptingRoleServiceSection）。下面的代码实例中阐述了这一点。

```
<section name="roleService" type="System.Web.Configuration.
ScriptingRoleServiceSection, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication" />
```

在 ASP.NET 应用程序的配置文件中只需要声明一次配置节处理程序即可，Web.config 文件和 ASP.NET 应用程序中的其他配置文件都自动继承在 Machine.config 文件中声明的配置处理程序。只有当创建用来处理自定义设置节的自定义节处理程序类时，用户才需要声明新的

节处理程序。

2. 配置节设置

配置节设置区域位于配置节处理程序声明区域之后，它包含实际的配置设置。默认情况下，在内部或者在某个根配置文件中，对于 configSections 区域中的每一个 section 和 sectionGroup 元素，都会有一个指定的配置节元素。这些配置节元素还可以包含子元素，这些子元素与其父元素由同一个节处理程序处理。例如，下面代码中的 pages 元素包含一个 controls 元素，该元素没有相应的节处理程序，因为它由 pages 节处理程序来处理。

```
<pages>
  <controls>
    <add tagPrefix="asp" namespace="System.Web.UI" assembly="System.Web.
    Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=
    31BF3856AD364E35"/>
    <add tagPrefix="asp" namespace="System.Web.UI.WebControls"
    assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
    PublicKeyToken=31BF3856AD364E35"/>
  </controls>
</pages>
```

6.3.3　在 Flex 中生成 Web.config

在网站的实际应用中，配置数据库连接字符串是 Web.config 文件最常用的功能之一，该功能主要在 connectionStrings 节配置。connectionStrings 元素为 ASP.NET 应用程序和 ASP.NET 功能指定数据库连接字符串（名称/值对的形式）的集合、会话、成员资格、个性化设置和角色管理器等功能，均依赖于存储在 connectionStrings 元素中的连接字符串。该节点有 3 个子元素，表 6-5 中列出了该节点 3 个子元素以及这 3 个子元素的说明。

表 6-5　connectionStrings 节子元素

元素	说明
Add	向连接字符串集合添加名称/值对形式的连接字符串
Clear	移除所有对继承的连接字符串的引用，仅允许那些由当前的 add 元素添加的连接字符串
Remove	从连接字符串集合中移除对继承的连接字符串的引用

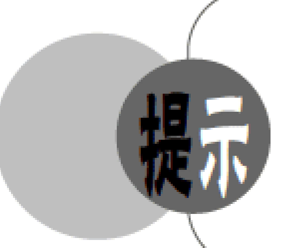
下面列举一个该节点的实例，代码如下所示。

```
<connectionStrings>
   <add  name="LocalSqlServer"  connectionString="data source=.\SQLEXPRESS;
   Integrated Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;
   User Instance=true"  providerName="System.Data.SqlClient"/>
</connectionStrings>
```

在 Flex Builder 3 开发环境中，为当前项目创建数据库应用时，会自动为项目创建 Web.config 文件，并在文件中存放相应的数据库连接字符串信息。下面将详细介绍如何在 Flex

中自动生成 Web.config 文件。具体步骤如下所示。

（1）选择 Data | Create Application from Database 命令，进入 Choose data source 窗口，如图 6-13 所示。



在进入 Choose data source 窗口前，要确保项目创建的时候，选择了应用程序服务器的类型，否则将无法打开选择数据源。该项目在“应用程序服务器类型”选项中选择了“ASP.NET”。

（2）单击 New…按钮，创建新的数据库连接。首先进入 Create connection profile 窗口，在这里设置 Name 为 connstr，Description 为“创建连接字符串”，并且可以启用 Auto-connect when the wizard is finished or when Data Source Explorer opens（向导完成或打开引用数据源的项目时自动连接）复选框，如图 6-14 所示。

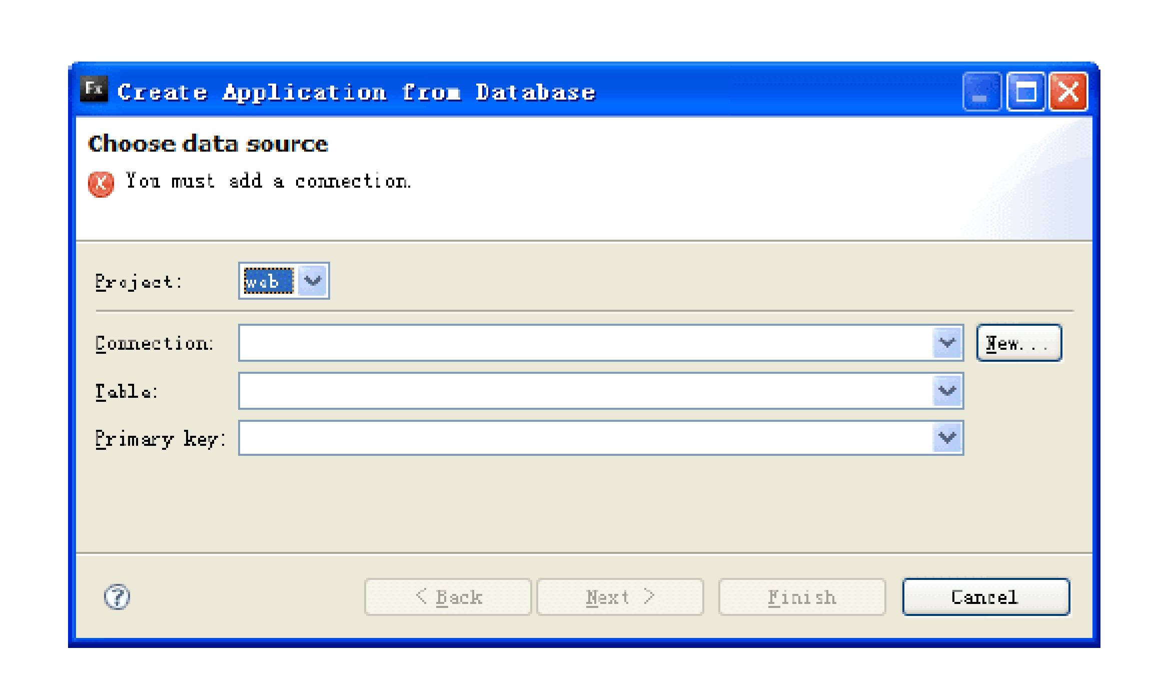




图 6-13　选择数据源

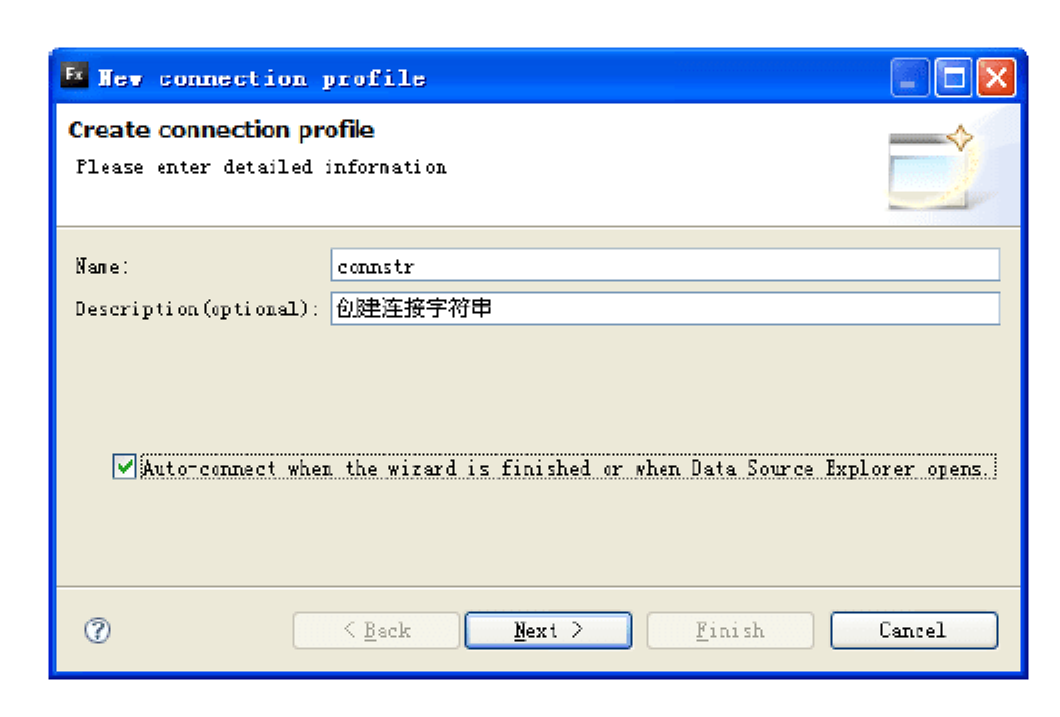


图 6-14　创建连接记录

（3）单击 Next 按钮，进入 Simple SQL Server Connection 窗口，创建一个 SQL Server 数据库的连接。在该窗口中，输入服务器、数据库名、SQL Server 登录账户名称和密码，如图 6-15 所示。

（4）配置完成后，单击 Test Connection 按钮，测试与数据库的连接，如果正确则显示 The connection was successful，如图 6-16 所示。

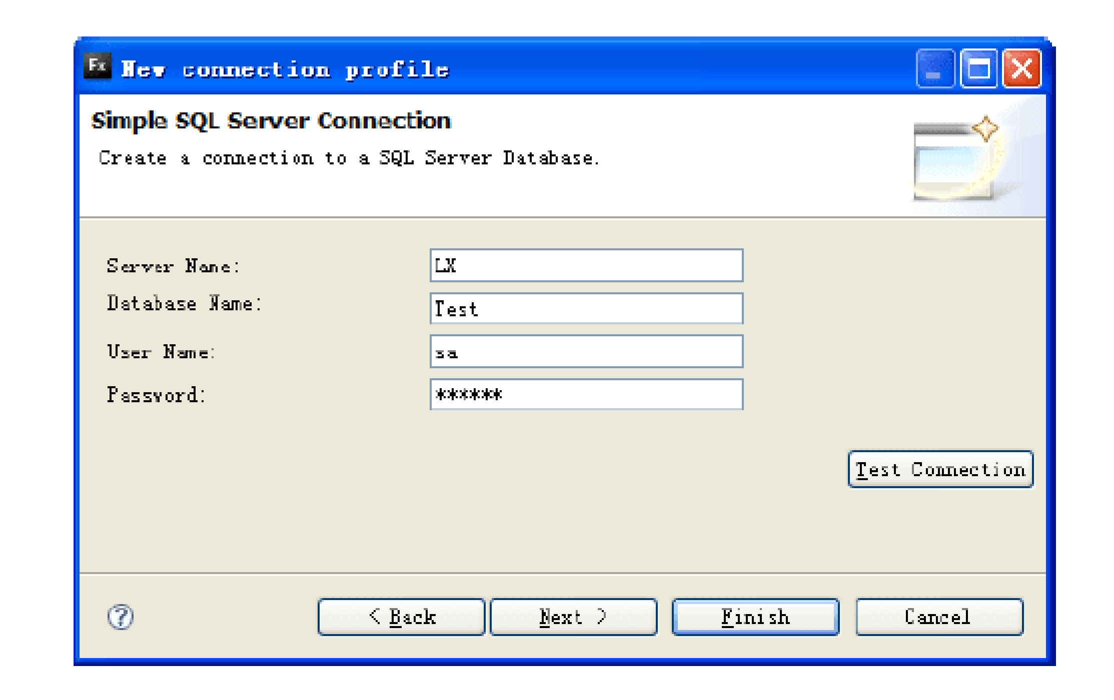


图 6-15　连接 SQL Server 数据库

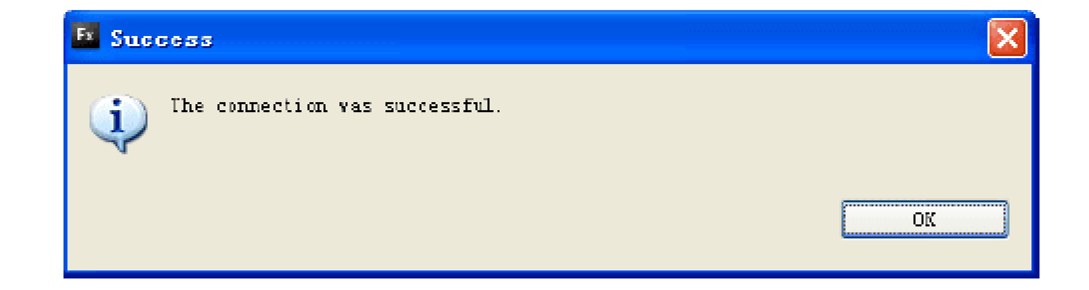


图 6-16　测试连接

（5）单击 Next 按钮，进入 Summary 窗口，在该窗口中，确认前面进行的设置，确认无误

后单击 Finish 按钮完成创建，如图 6-17 所示。

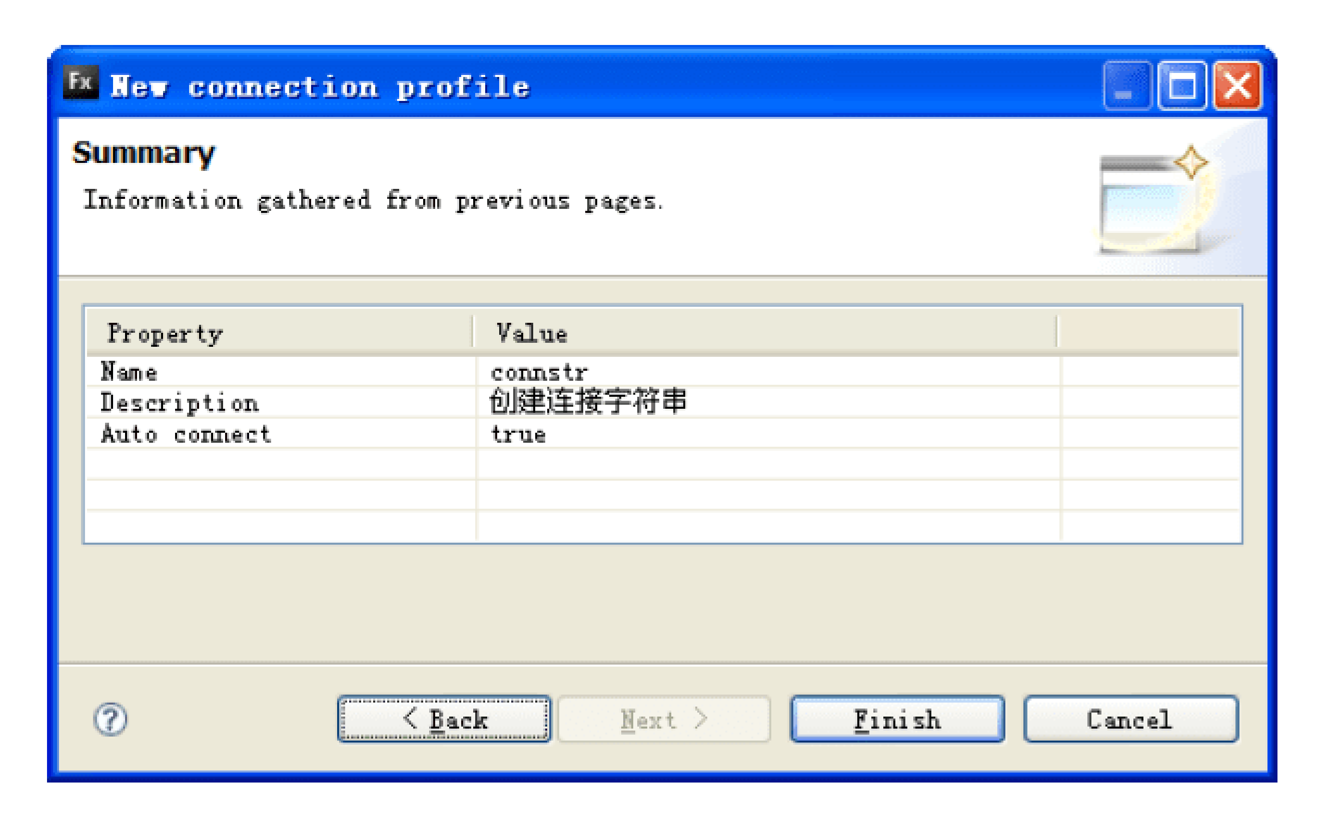


图 6-17　确认信息

（6）创建完成后，在存放项目文件的根文件夹下将会自动创建 Web.config 文件，该文件的具体内容如代码 6-14 所示。

代码 6.14　Web.config 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
   <appSettings/>
   <connectionStrings>
<add name="constr" connectionString="server=LX;database=TEST;uid=sa;
pwd=123456"  providerName="System.Data.SqlClient"/>
   </connectionStrings>
   <system.web>
      <!--
         The <authentication> section enables configuration
         of the security authentication mode used by
         ASP.NET to identify an incoming user.
           Please notice that the generated server side code does not have
           any
           specific authentication mechanism in place. We just use the default
           'Windows' setting to configure access.
           For more information please consult the documentation.
      -->
      <authentication mode="Windows"/>
      <!--
         The <customErrors> section enables configuration
         of what to do if/when an unhandled error occurs
         during the execution of a request. Specifically,
         it enables developers to configure html error pages
         to be displayed in place of a error stack trace.
      <customErrors mode="RemoteOnly" defaultRedirect= "GenericErrorPage.
```

```
        htm">
            <error statusCode="403" redirect="NoAccess.htm" />
            <error statusCode="404" redirect="FileNotFound.htm" />
        </customErrors>
        -->
         <trace enabled="true"/>
    </system.web>
</configuration>
```

第7章 ASP.NET 数据显示



内容摘要 | Abstract

在应用程序中经常会用到数据库来存取数据，几乎所有的应用程序都离不开数据的增加、删除、修改和查询操作，而这些操作往往是通过数据库来实现的。ASP.NET 提供了强大的数据库处理功能。在 ASP.NET 中处理数据的基本方法是通过 ADO.NET 中提供的数据访问类实现的。利用 ADO.NET 访问类，用户可以在程序中浏览、编辑各种数据库的数据。

另外，XML 具有自描述性、内容与显示相分离、可扩展性、独立于平台等特点。XML 还提供了一套跨平台、跨网络、跨程序语言的数据描述方式，已逐渐成为程序中数据表示及应用的数据交换标准格式。

ASP.NET 把 XML 作为应用程序数据存储和传输也是一种重要方法。将这两种技术结合起来可以开发出各种功能强大的系统。在本章中，我们将首先对 ADO.NET 和 XML 的基本概念进行介绍，然后详细讲解如何使用 ADO.NET 显示数据库数据，显示 XML 文件数据以及如何生成 XML 等。

学习目标 | Objective

- 了解 ADO.NET 命名空间
- 熟悉 ADO.NET 组件和对象的使用方法
- 了解 ListView 控件模板及数据显示方法
- 掌握 SqlDataSource 和 DataList 控件的使用方法
- 掌握 GridView 控件编辑数据的方法
- 熟悉 Repeater 控件显示数据的方法
- 掌握 4 种 XML 显示方法
- 掌握 XML 生成的两种方法

7.1 ADO.NET 概述

作为.NET Framework 框架的一部分，ADO.NET 提供了一组.NET 类，这些类不仅可以对各种数据源进行高效访问，还能够对数据进行复杂的操作和排序，而且形成了一个重要的框架，在这个框架中可以实现应用程序之间的通信和 XML Web 服务。

7.1.1　ADO.NET 命名空间

ADO.NET 提供了对 Microsoft SQL Server 等数据源，以及 OLE DB 和 XML 数据源的一致访问。数据共享使用者应用程序可以使用 ADO.NET 来连接到这些数据源，并检索、操作和更新数据。

ADO.NET 包含用于连接到数据库、执行命令和检索结果的.NET Framework 数据提供程序。可以直接处理检索到的结果，或将其放入 ADO.NET DataSet 对象，以便与来自多个源的数据或在层之间进行远程处理的数据组合在一起，以特殊方式向用户公开。ADO.NET DataSet 对象也可以独立于.NET Framework 数据供程序使用，以管理应用程序本地的数据或源自 XML 的数据。

ADO.NET 类在 System.Data.dll 中，并且与 System.Xml.dll 中的 XML 类集成。当编译使用 System.Data 命名空间的代码时，需要引用 System.Data.dll 和 System.Xml.dll。ADO.NET 命名空间主要在.NET Framework 的 System.Data 命名空间中，该命名空间主要包括以下几部分。

- **System.Data**

这个命名空间表示内存数据的类，由构成 ADO.NET 结构的类组成，该结构是托管应用程序的主要数据访问方法。ADO.NET 结构使生成的组件能够有效地管理来自多个数据源的数据。ADO.NET 还提供对分布式应用程序中的数据进行请求、更新和协调的工具。这些类独立于数据的源，所以无论数据来自 SQL Server、Access 还是 XML 文件，都可以使用相同的类。这些类中最为重要的是 DataSet 类。

- **System.Data.Common**

该命名空间包含由.NET Framework 数据提供程序共享的类。数据提供程序描述一个类的集合，这些类用于在托管空间中访问数据源，例如数据库。

- **System.Data.OleDb**

该命名空间构成兼容数据源的 OLE DB.NET Framework 数据提供程序的类。这些类能连接到 OLE DB 数据源，针对数据源执行命令并读取结果。因为用于 OLE DB 的.NET Framework 数据提供程序描述了用于访问托管空间中 OLE DB 数据源的类集合。使用 OleDbDataAdapter，可以填充驻留在内存中的 DataSet 类，该数据集可用于查询和更新数据源。

- **System.Data.SqlClient**

该命名空间构成 SQL Server.NET Framework 数据提供程序的类，提供程序允许连接到 SQL Server 2008、执行命令并读取结果。System.Data.SqlClient 命名空间与 System.Data.OleDb 命名空间类似，但对访问 SQL Server 2008 和更新版本进行了优化。

- **System.Data.SqlTypes**

该命名空间为 SQL Server 内的本机数据类型提供类。这些类提供了一种较之其他数据类型更安全、更快捷的方法。在可能丢失精度的情况下，在此命名空间中使用这些类有助于防止产生类型转换错误。由于其他数据类型隐式地与 SqlTypes 进行相互转换，所以在此命名空间内显式创建和使用对象将会使代码更快。例如，SQL Server 2008 中的 money 型列中的任意一个值都可以用 System.Data.SqlType.SqlMoney 类的一个实例来表示。

7.1.2　ADO.NET 组件

ADO.NET 包含两个核心组件，DataSet 和.NET Framework 数据提供程序。.NET Framework 数据提供程序包括 Connection 对象、Command 对象、DataReader 对象和 DataAdapter 对象，而 DataSet 包含 DataTable 对象集合和 DataRelation 对象集合。.NET Framework 数据提供程序用于连接数据源、执行 SQL 语句命令和检索数据。检索到的数据既可以直接处理，也可以放入 DataSet 对象中。.NET Framework 数据提供程序在 DataAdapter 对象和 DataSet 对象集合之间建立联系。ADO.NET 的对象模型如图 7-1 所示。

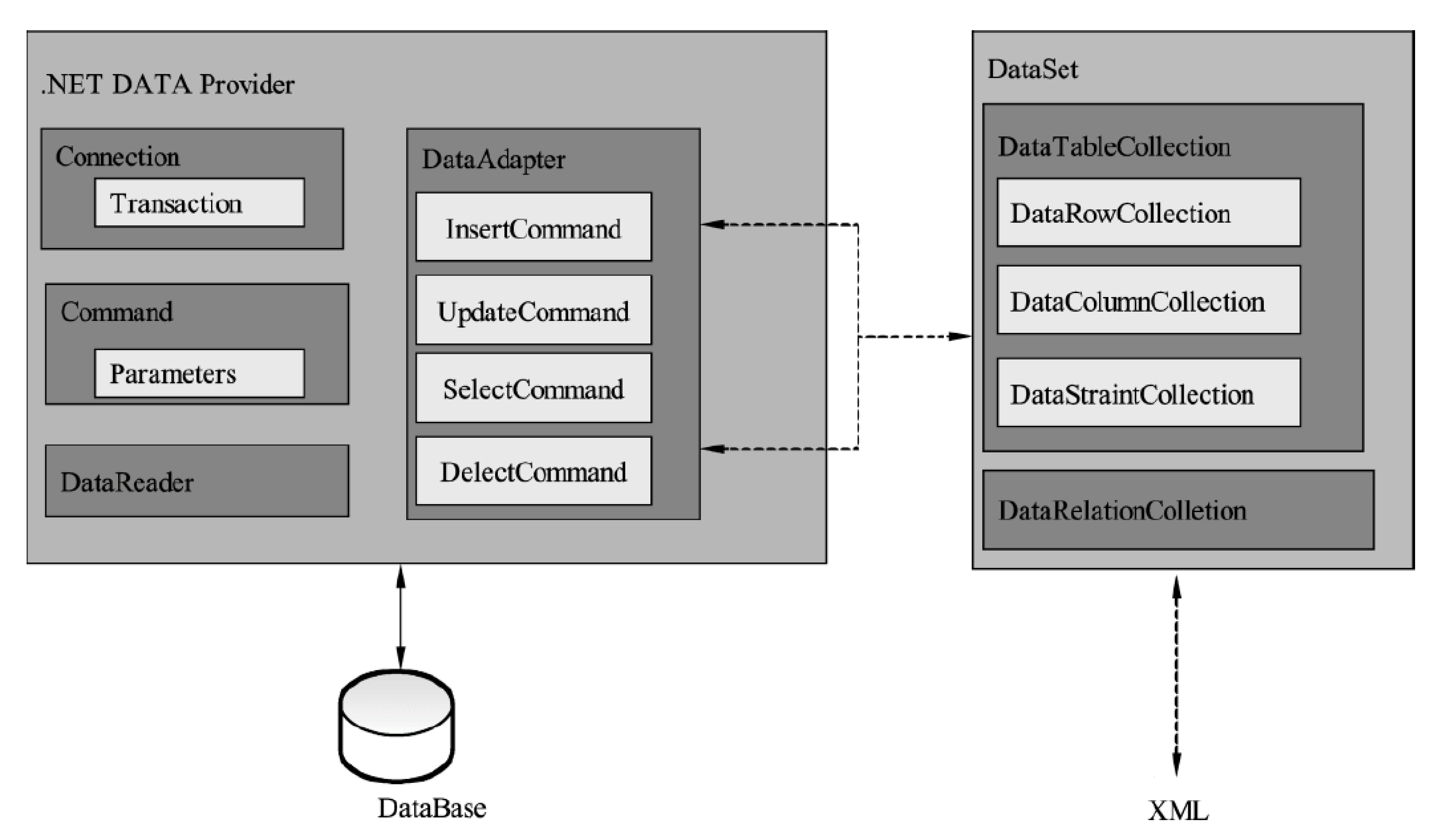


图 7-1　ADO.NET 对象模型

从图 7-1 中看到，在 ADO.NET 对象模型中主要有 5 个对象，分别是：Connection 对象、Command 对象、DataAdapter 对象、DataSet 对象以及 DataReader 对象。这些对象中负责建立链接和数据操作的部分称为数据操作组件，分别由 Connection 对象、Command 对象、DataAdapter 对象以及 DataReader 对象所组成。数据操作对象的最主要作用是当作 DataSet 对象以及数据源之间的桥梁，负责将数据源中的数据取出后写入 DataSet 对象中，以及将数据存回数据源的工作。

7.1.3　ADO.NET 对象

在了解 ADO.NET 所需的命名空间，并熟悉其组成结构之后，本节将详细介绍 ADO.NET 中与数据库有关的各种对象，像使用 Connection 对象建立连接，使用 Command 对象对象执行 SQL 命令并将查询结果保存到 DataSet 对象等。

1. Connection 对象

在对数据库进行操作之前，必须先定义一个数据库连接。因为，只有定义了这个连接字符串，才能够通过这个连接字符串将应用程序与数据库连接起来，用户才可以通过应用程序对数据库中的数据进行增加、删除、修改和查询等操作。Connection 对象主要处理对数据库的连接和管理数据库事务操作，是操作数据库的基础。

例如，如下的语句实例化了一个 SqlConnection 对象，创建到 SQL Server 服务器 zhht 中 schoolmis 数据库的连接，连接使用账户 sa，密码 123456。

```
SqlConnection conn = new SqlConnection("server=zhht;uid=sa;
pwd=123456;database=schoolmis;");
```

创建了 Connection 对象之后，需要打开连接才能连接到数据库，为此可以调用 Connection 对象的 Open()方法。当使用 Connection 对象之后，最好关闭并释放 Connection 对象，来缓解数据库服务器的压力。可以调用 Connection 对象的 Close()方法关闭数据库连接。其打开和关闭语句如下：

```
conn.Open();     //打开数据库连接
conn.Close();    //关闭数据库连接
```

2. Command 对象

在建立到数据库的连接后，需要使用一个数据库操作对象来实现。一个数据库操作命令可以用 SQL 语句来表达，包括执行选择查询（SELECT 语句）来返回记录集合，执行更新查询（UPDATE 语句）来执行更新记录，执行删除查询（DELETE 语句）来删除记录等。Command 命令也可以传递参数并返回值，同时 Command 命令也可以被明确地定界，或调用数据库中的存储过程。

例如，如下的语句演示了使用 Command 对象在 conn 连接的数据库中查询表 books 的内容。

```
string selectSQL = "select * from books";
SqlCommand cmd = new SqlCommand(selectSQL, conn);
```

3. DataReader 对象

DataReader 对象是用来读取数据库的最简单方式，它只能读取、不能写入，并且是从头至尾往下读。因此，DataReader 对象只能只读某条数据，但它占用内存小、速度快。每次在内存中的数据只有一行，所以使用 DataReader 可提高应用程序的性能并减少系统开销。

要创建 DataReader 对象，首先建立 Command 对象，再确认执行的 SQL 语句，最后用 ExecuteReader()方法返回一个 DataReader 对象。例如，如下的代码段演示了创建 DataReader 对象实例 myReader，并使用它获取 bkname 和 bkpub 列信息的过程。

```
SqlDataReader myReader=cmd.ExecuteReader();
if (myReader.Read())
```

```
{
    bookname = myReader["bkname"].ToString().Trim();
    bookpub = myReader["bkpub"].ToString().Trim();
}
```

4．DataAdapter 对象

DataAdapter 对象可以用于检索和更新数据库，或从数据源中获取数据，填充 DataSet 对象中的表，以及对 DataSet 对象进行更改，提交回数据源。该对象是 DataSet 对象和数据库之间的桥梁。DataAdapter 对象是 ADO.NET 托管提供程序的组成部分，该对象是可配置的，允许用户指定将哪些数据移入或移出数据集。经常采取的形式是对 SQL 语句或存储过程调用，这些语句或存储过程被调用时即可实现对数据库的读写。

ADO.NET 允许以两种方式从数据库中检索数据：一种是使用 DataReader 对象，它只读数据流；第二种是使用 DataAdapter 对象，该对象与 DataSet 紧密配合创建数据的内存表示。下面的语句创建了 DataAdapter 对象 myAdapter，并指定了 SQL 语句 selectSQL 和数据库连接 conn。

```
string selectSQL = " select * from books ";
SqlDataAdapter myAdapter = new SqlDataAdapter(selectSQL, conn);
```

5．DataSet 对象

DataSet 对象与 DataReader 对象一样提供一个记录集，但不像使用 DataReader 对象那样有一定限制。DataSet 对象中可以包括一个或多个 DataTable，并且还包括 DataTable 之间的约束等关系。

如下的语句创建 DataSet 对象的实例 mydataset。

```
DataSet mydataset = new DataSet();
```

向 mydataset 中填充数据并指定别名为 bookset，使用以下语句。

```
myAdapter.Fill(mydataset, "bookset");
```

6．DataTable 对象

DataTable 对象也是 ADO.NET 库中的核心对象之一，它提供强大的功能和灵活性，数据的处理大都在 DataTable 中进行处理。

DataTable 是元数据和数据的集合，其中，元数据为 DataColumn 对象和 Constraint 对象的集合描述，而数据则包含在 DataRow 对象的集合中。DataTable 可以独立存在，也可以是 DataSet 的一部分。与 ADO Recordset 对象不同，DataTable 是一个被动对象。DataAdapter 对象、XmlDataDocument 对象和用户代码可操作 DataTable 对象。DataTable 不知道自己的数据来自何处。这些数据可以来自多个源。

创建 DataTable 对象的方法如下：

```
DataTable myda = new DataTable("DataTableName");
```

下面的代码演示了从 DataSet 中获取 DataTable 的方法。

```
SqlDataAdapter myDataAdapter = new SqlDataAdapter();
DataSet myDataSet = new DataSet();
myDataAdapter.Fill(myDataSet, "bookset");
DataTable myDataTable1 = myDataSet.Tables[0];          //第一种使用索引获取
DataTable myDataTable2 = myDataSet.Tables["bookset"];  //第二种使用名称获取
```

7. Parameter 对象

Parameter 对象集合关联着一个命令（Command）对象。Parameter 对象代表 SQL Server 存储过程的参数或查询中的参数。在 Command 对象中，有多个 Parameter 子对象可以用来存储参数，这些 Parameter 对象都收集在 Parameters 集合中。该集合中包括 Count 属性、Append 方法、DELETE 方法、Refresh 方法和 Item 方法。Parameter 对象负责记录程序中要传递参数的相关属性，Parameter 对象提供了 Name 属性、Value 属性、Type 属性和 Attribute 属性等。

若要实现查询，SQL 查询语句需要赋值给一个 SqlCommand 对象的只是一个简单的字符串。当想过滤一个查询，可以动态地绑定字符串，但是本来不想这样做，下面是一个过滤查询的示例。

```
SqlCommand cmd = new SqlCommand("select * from News where newsCity ='" +
CityName + "'");
```

上面的例子可以使用动态创建字符串的方法替代，使用 Parameters 集合。任何放置在 parameter 中的东西都将被作为字段数据处理，而不是 SQL 语句的一部分。这样就让应用程序更加安全。使用参数化查询的步骤如下所示。

- 使用 parameters 构建 SqlCommand 命令字符串。
- 声明 SqlParameter 对象，将适当的值赋给它。
- 将 SqlParameter 对象赋值给 SqlCommand 对象的 Parameters 属性。

首先为 Parameters 准备 SqlCommand 对象，在 SQL 查询中使用 Parameters 的第一步是创建包含参数占位符的对象字符串。这些占位符在 SqlCommand 执行的时候填充实际的参数值。Parameter 的正确的语法是使用一个“@”符号作为参数名的前缀，如下所示。

```
SqlCommand cmd = new SqlCommand("select * from News where newsCity =
@CityName",con);
```

在 SQL 语句中的每一个参数必须被定义，这是 SqlParameter 类型的需要。代码中必须为每一个在 SqlCommand 对象的 SQL 命令中的参数定义一个 SqlParameter 实体。下面的代码实现@CityName 参数定义。

```
SqlParameter para  = new SqlParameter();
para.ParameterName = "@CityName";
para.Value=CityName;
```

对于每一个定义在 SqlCommand 对象中的 SQL 命令字符串参数，必须定义一个

SqlParameter 实体。同样将 SqlParameter 实体赋值给 SqlComand 对象的 Parameters 属性的方式，让 SqlCommand 对象知道 SqlParameter。实现代码如下所示。

```
cmd.Parameters.Add(para);
```

7.2 数据显示控件

数据显示控件是 ASP.NET 中的一种服务器控件，主要作用是呈现返回的结果集。ASP.NET 3.5 是对之前版本的升级，对旧版本中数据显示上的不足进行了改进，主要体现在数据访问和显示的简单化、智能化、多样化以及提高数据显示的效率和性能等方面。本节将介绍 ListView 控件、DataList 控件、GridView 控件和 Repeater 控件的使用方法。

7.2.1 ListView 控件

ListView 控件是 ASP.NET 3.5 中的新增数据显示控件，它结合了 Repeater 与 GridView 控件，实现添加、删除等功能。同时还可以像 Repeater 一样灵活地控制页面布局。该控件包含了很多新的模板，例如 GroupTemplate 等新增的模板，可以方便地分组显示数据。ListView 控件本身并不提供分页功能，但可以使用 ASP.NET 3.5 中另一个新增的控件 DataPager 实现分页功能。把分页的功能单独放到另一个控件里，能够给编程带来很大的方便，比如说可以让别的控件使用它，也可以把它放到页面的任何地方。实质上，DataPager 控件就是一个扩展 ListView 分页功能的控件。

1. ListView 控件模板

ListView 控件是一个非常灵活的控件，这是因为 ListView 控件提供了很多模板。通过定义 ListView 的模板几乎可以实现任意一种数据显现方式。从这一点来说，ListView 控件和 DataList 控件、Repeater 控件很相似，与这些控件不同的是，ListView 控件允许用户编辑、插入和删除数据，以及对数据进行排序和分页。下面对 ListView 控件可用模板介绍如下。

- **LayoutTemplate**

标识定义控件主要布局的根模板。它包含一个占位符对象，如表行 tr、div 或 span 元素。此元素将由 ItemTemplate 模板或 GroupTemplate 模板中定义的内容替换，它还可能包含一个 DataPager 对象。通过创建 LayoutTemplate 模板，可以定义 ListView 控件的主要根布局。LayoutTemplate 必须包含一个充当数据占位符的控件。

- **ItemTemplate**

标识要为各个项显示的数据绑定内容，该模板在前面已经介绍过，在此就不再重复。

- **GroupTemplate**

标识组布局的内容。它包含一个占位符对象，例如表行 td、div 或 span。该对象将由其他模板，例如 ItemTemplate 和 EmptyItemTemplate 模板中定义的内容替换。使用 GroupTemplate 模板，可以选择对 ListView 控件中的项进行分组。对项分组通常是为了创建平铺的表布局。

在平铺的表布局中，各个项将在行中重复 GroupItemCount 属性指定的次数。

- **EmptyDataTemplate**

标识在数据源未返回数据时要呈现的内容。

- **SelectedItemTemplate**

标识为区分所选数据项与显示的其他项，而为该所选项呈现的内容。

- **AlternatingItemTemplate**

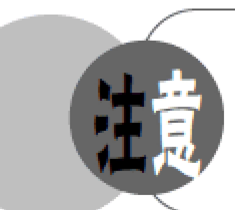
标识为便于区分连续项，而为交替项呈现的内容。

- **EditItemTemplate**

标识要在编辑项时呈现的内容。对于正在编辑的数据项，将呈现 EditItemTemplate 模板以替代 ItemTemplate 模板。

- **InsertItemTemplate**

标识要在插入项时呈现的内容。将在 ListView 控件显示的项的开始或末尾处呈现 InsertItemTemplate 模板，以替代 ItemTemplate 模板。通过使用 ListView 控件的 InsertItemPosition 属性，可以指定 InsertItemTemplate 模板的呈现位置。



为能使 ListView 控件显示数据，必须为 ListView 控件创建模板，而且 LayoutTemplate 和 ItemTemplate 是必需的，其他所有模板都是可选的。

2．使用 ListView 控件显示数据

在 ListView 控件中创建 LayoutTemplate 模板，可以定义 ListView 控件显示数据的布局。LayoutTemplate 必须包含一个充当数据占位符的控件。例如，该布局模板可以包含 ASP.NET Table、Panel 或 Label 控件，还可以包含 runat 属性设置为 server 的 table、div 或 span 元素。这些控件将包含 ItemTemplate 模板所定义的每个项的输出，如果为显示数据分组的话，可以在 GroupTemplate 模板定义的内容中对这些输出进行分组。

使用 GroupTemplate 模板，选择对 ListView 控件中的项进行分组。对项分组通常是为了创建平铺的表布局。在平铺的表布局中，各个项将在行中重复 GroupItemCount 属性指定的次数。为创建平铺的表布局，布局模板可以包含 ASP.NET Table 控件以及将 runat 属性设置为 server 的 HTML table 元素。随后，组模板可以包含 ASP.NET TableRow 控件或 HTML tr 元素。而项模板可以包含 ASP.NET TableCell 控件中的各个控件。

下面创建一个 ListView 显示数据的实例，该实例使用到了两种必需的模板，还有一个分组模板和一个交替模板，示例具体开发步骤如下所示。

（1）首先添加一个 ListView 控件到页面。切换至【源】视图，在 ListView 控件的源代码处，添加一个 LayoutTemPlate 模板。该步骤主要使用代码 7.1 所示的内容。

代码 7.1　设置 ListView 控件的 LayoutTemPlate 模板

```
<LayoutTemplate>
    <table runat="server">
        <tr runat="server">
```

```
            <td runat="server">
                <table ID="groupPlaceholderContainer" runat="server" border="1"
                    style="background-color: #FFFFFF;border-collapse:
                    collapse;border-color: #999999;border-width:1px;
                    font-family: Verdana, Arial, Helvetica, sans-serif;">
                    <tr ID="groupPlaceholder" runat="server">
                    </tr>
                </table>
            </td>
        </tr>
    </table>
</LayoutTemplate>
```

（2）添加一个 GroupTemplate 模板，在该模板中编写如代码 7.2 所示的内容。

代码 7.2　设置 ListView 控件的 GroupTemplate 模板

```
<GroupTemplate>
    <tr ID="itemPlaceholderContainer" runat="server">
        <td ID="itemPlaceholder" runat="server">
        </td>
    </tr>
</GroupTemplate>
```

（3）添加一个显示项的 ItemTempLate 模板，该模板用于显示数据，在该模板中我们使用 Eval()方法绑定数据，该模板的布局如代码 7.3 所示。

代码 7.3　设置 ListView 控件的 ItemTempLate 模板

```
<ItemTemplate>
    <td runat="server" style="background-color:#EfEfEf;color: #000000;">
        课程编号：<asp:Label ID="课程编号 Label" runat="server" Text='<%# Eval
        ("课程编号") %>' />
    <br /> 课程名称：<asp:Label ID="课程名称 Label" runat="server" Text='<%# Eval
    ("课程名称") %>' />
    <br /> 班级编号：<asp:Label ID="班级编号 Label" runat="server" Text='<%# Eval
    ("班级编号") %>' />
    <br />教师：<asp:Label ID="教师 Label" runat="server" Text='<%# Eval("教
    师") %>' />
    <br />开课系别：<asp:Label ID="开课系别 Label" runat="server" Text='<%# Eval
    ("开课系别") %>' />
    <br /> </td>
</ItemTemplate>
```

（4）添加一个交替显示模板，在该模板中编写如代码 7.4 所示的代码。

代码 7.4　设置 ListView 控件的 AlternatingItemTemplate 模板

```
<AlternatingItemTemplate>
```

```
            <td runat="server">
              课程编号: <asp:Label ID="课程编号 Label" runat="server" Text='<%# Eval
              ("课程编号") %>' />
            <br /> 课程名称: <asp:Label ID="课程名称 Label" runat="server" Text='<%# Eval
            ("课程名称") %>' />
            <br />班级编号: <asp:Label ID="班级编号 Label" runat="server" Text='<%# Eval
            ("班级编号") %>' />
            <br />教师: <asp:Label ID="教师 Label" runat="server" Text='<%# Eval("教
            师") %>' />
            <br />开课系别: <asp:Label ID="开课系别 Label" runat="server" Text='<%# Eval
            ("开课系别") %>' />
            <br /> </td>
        </AlternatingItemTemplate>
```

（5）此时，该实例就基本完成了，在该控件【属性】窗口设置其 GroupItemCount 值为 3。然后再为其添加一个数据源，运行该实例，页面效果如图 7-2 所示。

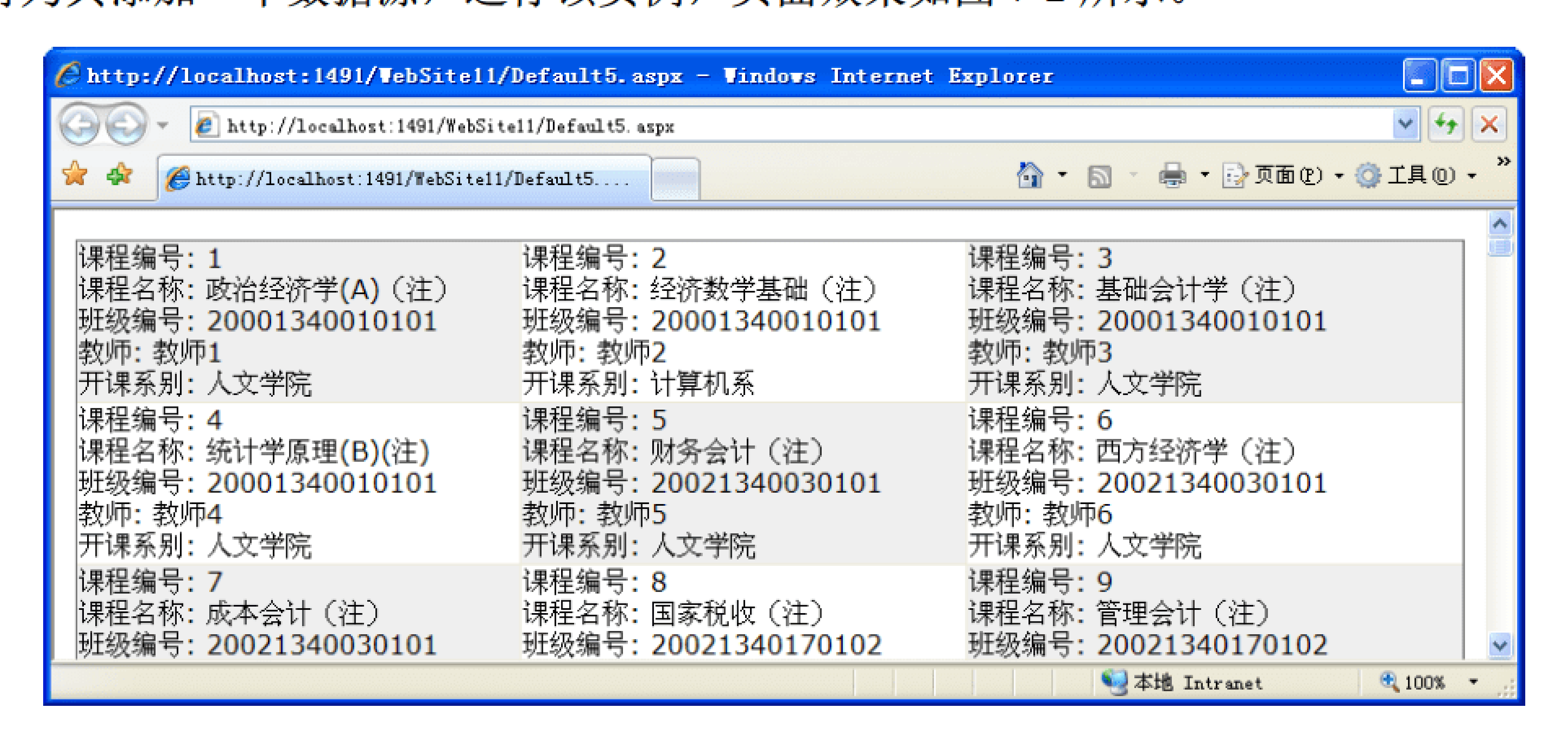


图 7-2　页面最终效果图

3．使用 ListView 控件分页显示数据

如果想为 ListView 控件增加分页功能的话，那么就需要使用 DataPager 控件了。ASP.NET 3.5 新增的 DataPager 控件主要用于 ListView 控件的分页，是新增的分页控件，可以方便、高效地进行分页。这个分页控件是一个独立的控件，可以把它放到页面的任何位置，然后使其连到 ListView 控件就可以实现分页。

为了使 DataPager 控件显示导航控件，必须向该控件添加页导航字段，该控件共包含如下 3 个字段。

- **NextPreviousPagerField**　用户能够逐页浏览页面，或跳到第一页或最后一页。
- **NumericPagerField**　用户能够按照页码选择页面。
- **TemplatePagerField**　可以创建自定义分页布局。

例如，将代码 7.5 添加到页面中的任何位置都可以实现 ListView 的分页功能。

代码 7.5　DataPager 控件实现 ListView 分页

```
<asp:DataPager ID="Pager" runat="server"  PagedControlID="ListView1"
PageSize="5" >
    <Fields>
        <asp:numericpagerfield ButtonCount="5" NextPageText=""
        PreviousPageText="" />
        <asp:nextpreviouspagerfield FirstPageText="First" LastPageText="Last"
            NextPageText="Next" PreviousPageText="Previous" />
    </Fields>
</asp:DataPager>
```

如代码 7.5 所示，通过设置 DataPage 控件的 Fields，可以达到手动分页布局的目的。另外有一个关键点，需要把 DataPager 控件的 PagedControlID 属性设置为 ListView 的 ID。也可以把 DataPager 控件放到布局模板中。DataPager 控件没有分页事件，也没有 SelectedPageIndex 属性。运行程序，结果如图 7-3 所示。

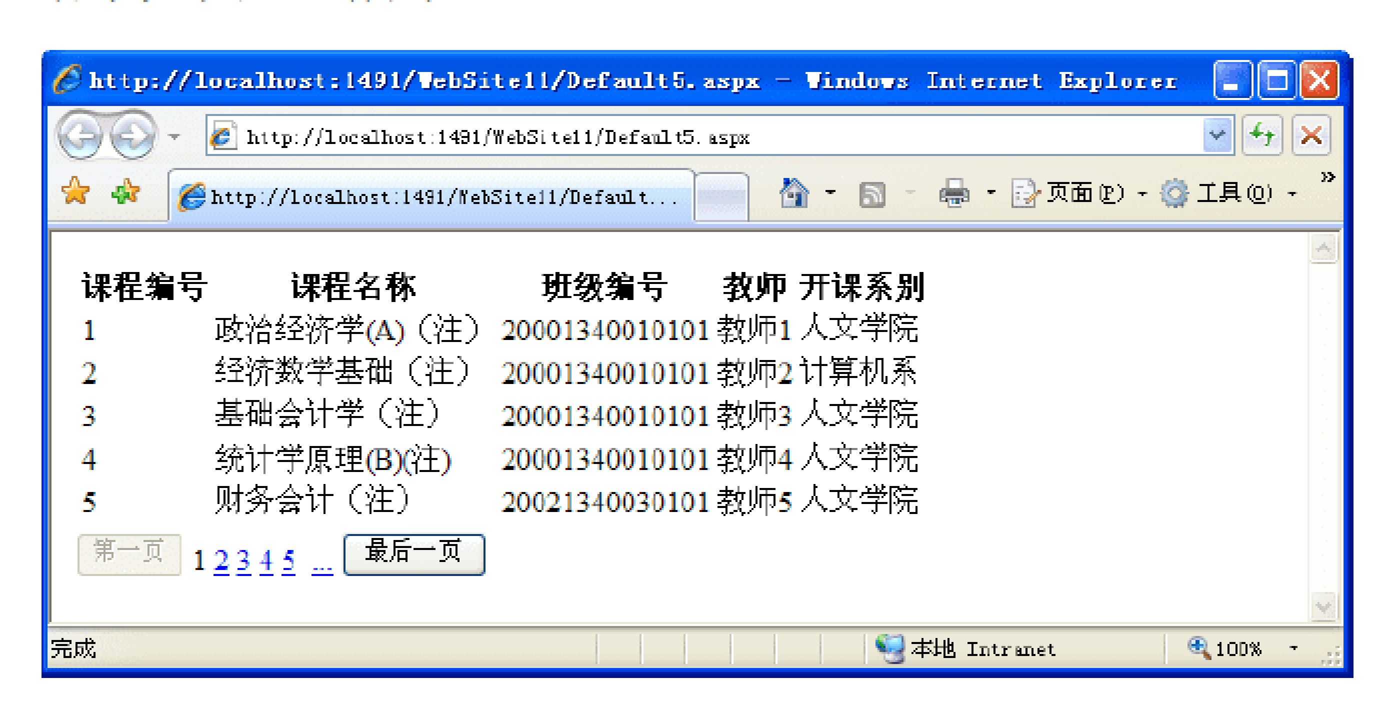


图 7-3　运行结果

7.2.2　DataList 控件

DataList 是一个常用的数据控件，可以自定义模板，显示数据比较灵活。通过该控件以表的形式显现数据，可以使用不同的布局来显示数据记录，例如，将数据记录排成列或行的形式。可以对 DataList 控件进行配置，使用户能够编辑或删除表中的记录。DataList 提供了 7 种模板，ItemTemplate、AlternatingItemTemplate、SelectedItemTemplate、EditItemTemplate、HeaderTemplate、FooterTemplate 和 SeparatorTemplate。

下面通过一个实例说明 DataList 控件的使用，实例通过使用 DataList 控件显示数据，使用的数据源是 SqlDataSource。具体步骤如下所示。

（1）首先在 ASP.NET 页面上添加一个 SqlDataSource 1，然后右击该控件选择【配置数据源】命令打开【配置数据源】对话框，如图 7-4 所示。

（2）这是配置数据源的一个向导，单击【新建连接】按钮弹出【添加连接】对话框，在这里配置 SQL Server 服务器的名称、登录信息和数据库，如图 7-5 所示。

（3）单击【确定】按钮后返回。再单击【配置数据源】对话框中的【下一步】按钮，弹

出向导保存连接字符串的对话框，启用该复选框，如图 7-6 所示。

图 7-4 【配置数据源】对话框

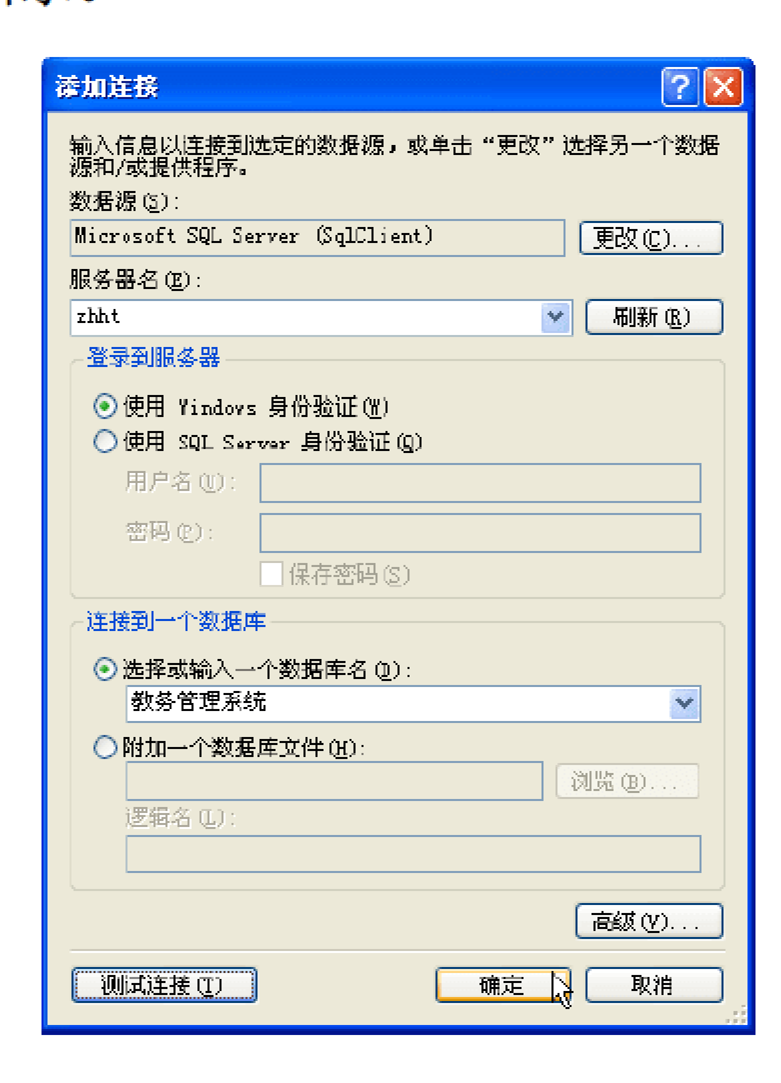


图 7-5 【添加连接】对话框

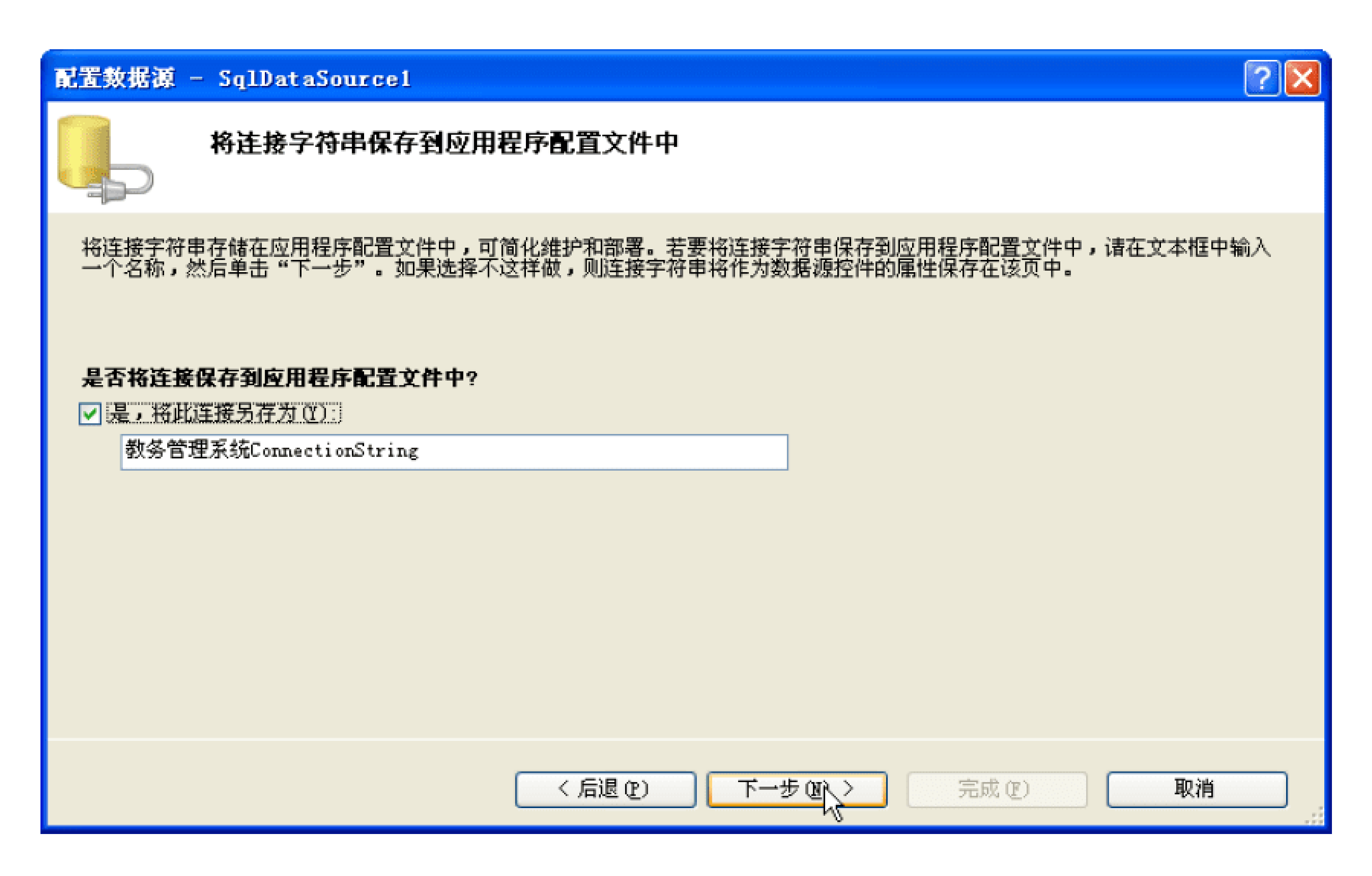


图 7-6　保存连接

（4）然后单击【下一步】按钮，弹出【配置 Select 语句】对话框，选择【指定来自表或视图的列】单选按钮，在【名称】下拉列表框中选择"学生信息"，并选择所有的列，如图 7-7 所示，在该对话框中可以为其添加 where 子句和分组子句，还可以为其添加 INSEET、UPDATE 和 DELETE 语句。

（5）单击【下一步】按钮，弹出【测试查询】对话框。在该对话框中单击【测试查询】按钮，如果测试成功数据会显示出来，如图 7-8 所示。

（6）最后单击【完成】按钮，就为该数据源控件配置好了数据。然后在 ASP.NET 页面上添加一个 DataList 控件。

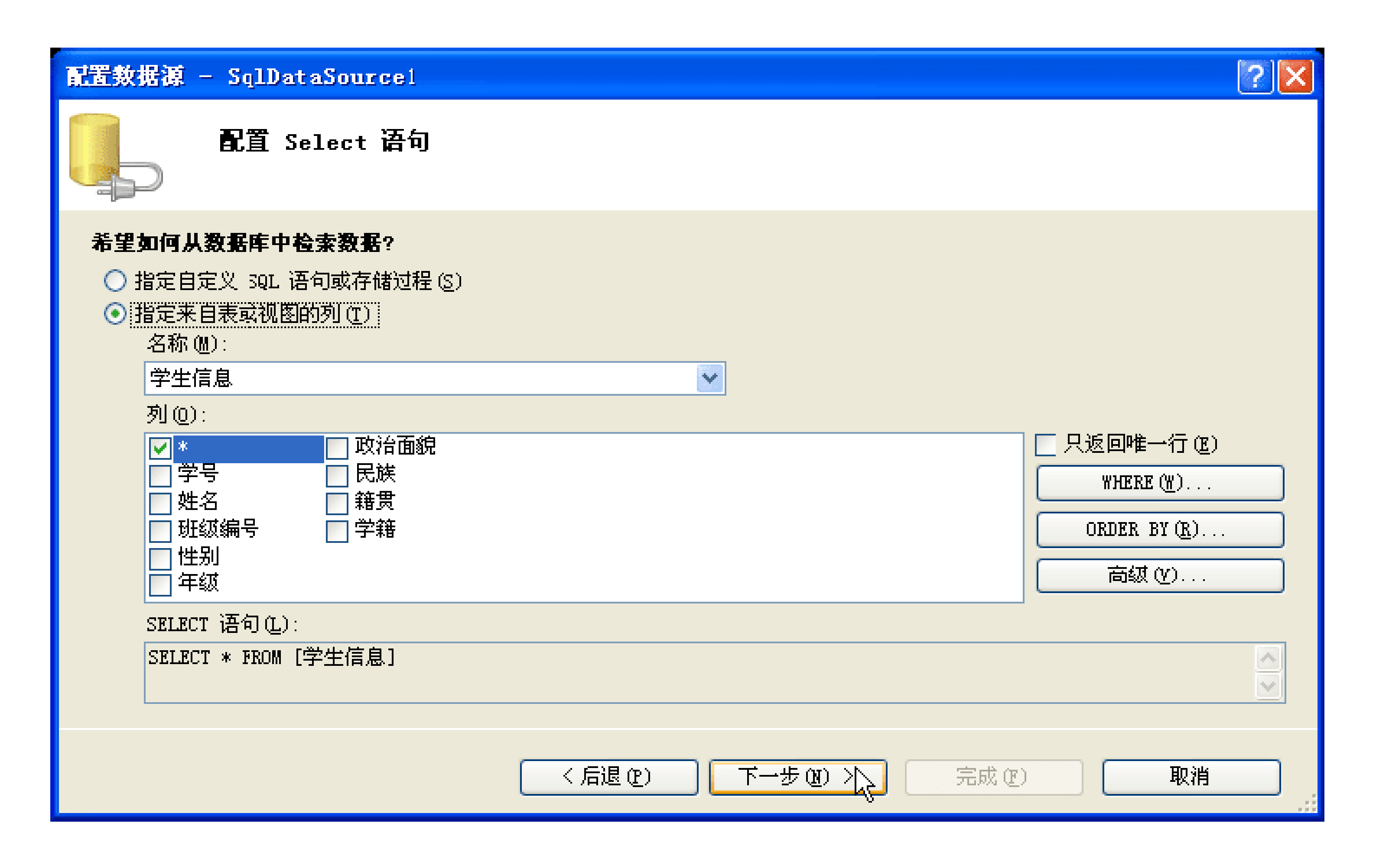


图 7-7 【配置 Select 语句】对话框

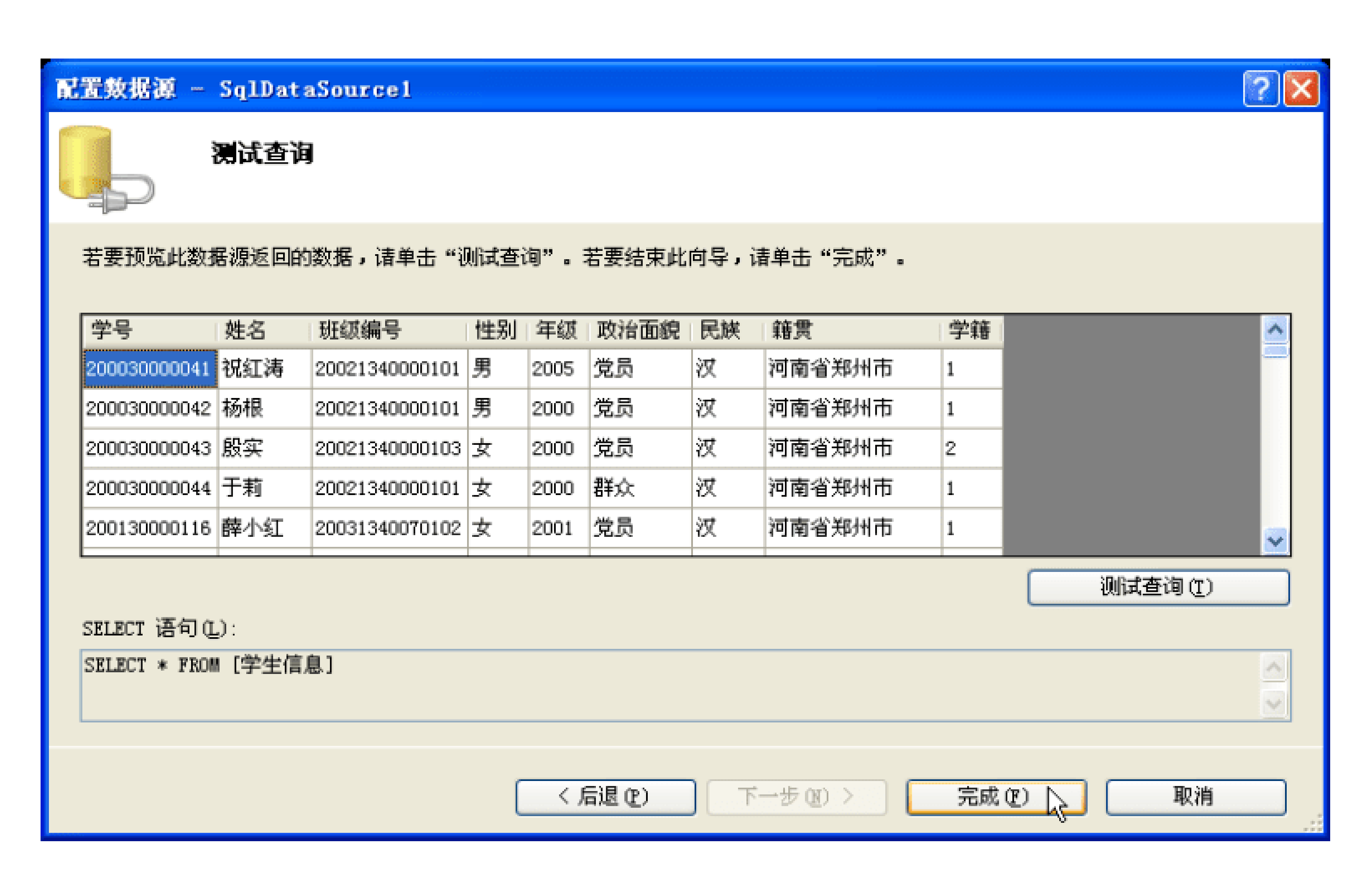


图 7-8 【测试查询】对话框

（7）在 DataList 控件中创建一个 HeaderTemplate 模板，在该模板中填写文字“DataList 的模板演示”。添加模板内容的代码如代码 7.6 所示。

代码 7.6 设置 DataList 控件 HeaderTemplate 模板

```
<HeaderTemplate>
    <h3>DataList 的模板演示</h3>
</HeaderTemplate>
<HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White">
</HeaderStyle>
```

（8）在 DataList 控件中再添加一个 ItemTemplate 模板，在该模板内首先添加一个“学号”和“姓名”文字，再添加数据的绑定。然后添加一个 LinkButton 按钮，并命名 CommandName 值为 select，Text 值为“更多”。代码 7.7 所示是这步操作的代码。

代码 7.7　设置 DataList 控件 ItemTemplate 模板

```
<ItemTemplate>
   学号: <%# DataBinder.Eval(Container.DataItem,"学号")%>
   姓名: <%# DataBinder.Eval(Container.DataItem,"姓名")%>
   <asp:LinkButton ID="btnSelect" CommandName="select" runat=
   "server"><img src="zoom.png"  />更多</asp:LinkButton><br>
</ItemTemplate>
<ItemStyle VerticalAlign="Top" BackColor="#EFF3FB"></ItemStyle>
```

（9）再添加一个 AlternatingItemTemplate 模板，在该模板内添加的内容和 ItemTemplate 模板的内容基本相同，如代码 7.8 所示。

代码 7.8　设置 DataList 控件 AlternatingItemTemplate 模板

```
<AlternatingItemTemplate>
   <span style="color: Red">学号: <%# DataBinder.Eval(Container.
   DataItem,"学号")%>
      姓名: <%# DataBinder.Eval(Container.DataItem,"姓名")%>
   </span>
   <asp:LinkButton ID="btnSelect1" CommandName="select" runat=
   "server"><img src="zoom.png" />更多</asp:LinkButton><br>
</AlternatingItemTemplate>
<AlternatingItemStyle BackColor="White" />
```

（10）再添加一个 SelectedItemTemplate 模板，该模板主要用于当用户选择时，显示详细信息的主要代码，如代码 7.9 所示。

代码 7.9　设置 DataList 控件 SelectedItemTemplate 模板

```
<SelectedItemTemplate>
   学号: <%# DataBinder.Eval(Container.DataItem,"学号")%>
   <asp:LinkButton ID="LinkButton4" CommandName="btnSelect"
   runat="server"> 返回</asp:LinkButton><br />
   姓名: <%# DataBinder.Eval(Container.DataItem,"姓名") %><br />
   班级编号: <%# DataBinder.Eval(Container.DataItem,"班级编号") %><br />
   性别: <%# DataBinder.Eval(Container.DataItem,"性别") %><br />
   民族: <%# DataBinder.Eval(Container.DataItem,"民族") %><br />
</SelectedItemTemplate>
<SelectedItemStyle BackColor="#D1DDF1" Font-Bold="True" ForeColor=
"#333333"></SelectedItemStyle>
```

（11）接下来，添加一个 FooterTemplate 模板，在该模板内显示了当天的日期。模板的最终布局如代码 7.10 所示。

代码 7.10　设置 DataList 控件 FooterTemplate 模板

```
<FooterTemplate>
```

```
    今天是: <%# DateTime.Now.ToLongDateString() %>
</FooterTemplate>
<FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
```

（12）单击 DataList 控件右上角的智能按钮，在【选择数据源】下拉列表框中选择 SqlDataSource1。完成数据源的配置，此时会弹出如图 7-9 所示的提示对话框，单击【否】按钮。

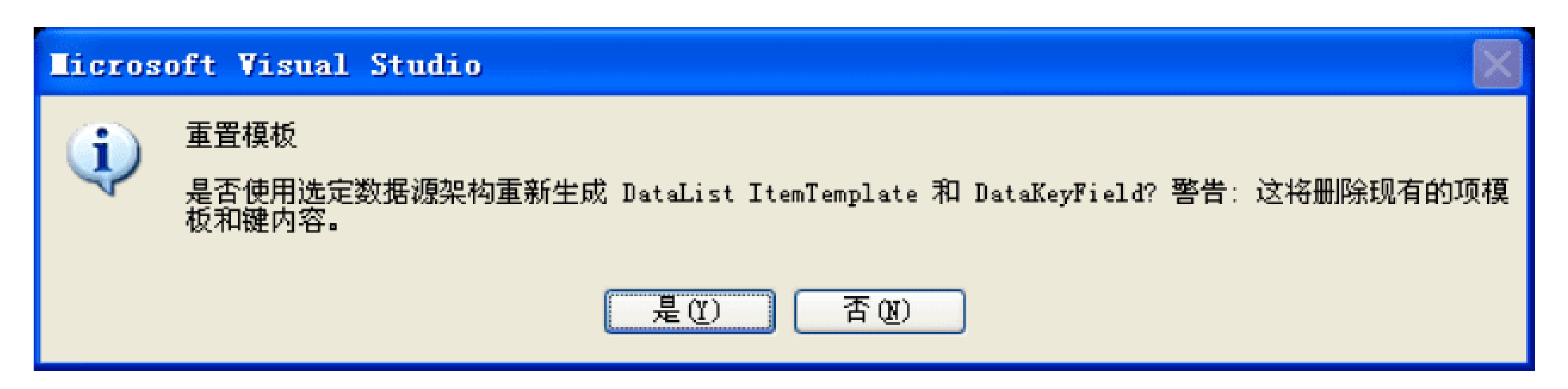


图 7-9　Microsoft Visual Studio 对话框

（13）现在编写后台代码。在 DataList 控件的【事件】窗口中双击 SelectedIndexChanged，进入后台代码编写窗口，在此事件处理程序中编写如代码 7.11 所示的内容。

代码 7.11　处理 SelectedIndexChanged 事件

```
protected void DataList1_SelectedIndexChanged(object sender, EventArgs e)
{
    DataList1.DataBind();
}
```

（14）在 DataList 的【事件】窗口中双击 ItemCommand。然后进入后台代码编写窗口，在此处编写如代码 7.12 所示的内容。

代码 7.12　处理 ItemCommand 事件

```
protected void DataList1_ItemCommand(object source,
DataListCommandEventArgs e)
{
    if (e.CommandName == "btnSelect")
    {
        DataList1.SelectedIndex = -1;
        DataList1.DataBind();
    }
}
```

至此，整个实例就设计完成。运行程序，当单击【更多】按钮时，会在“学号”的下方显示学生的详细资料，页面效果如图 7-10 所示。

7.2.3　GridView 控件

GridView 控件用来在表中显示数据源的值。每列表示一个字段，而每行表示一条记录。

GridView 控件提供了很多内置功能，使程序员写很少的代码或者是不写代码，就能设计出列出数据的表并且可以进行编辑、多选、排序和删除等。还可以通过设置该控件的外观属性，使其呈现出不同风格的外观。

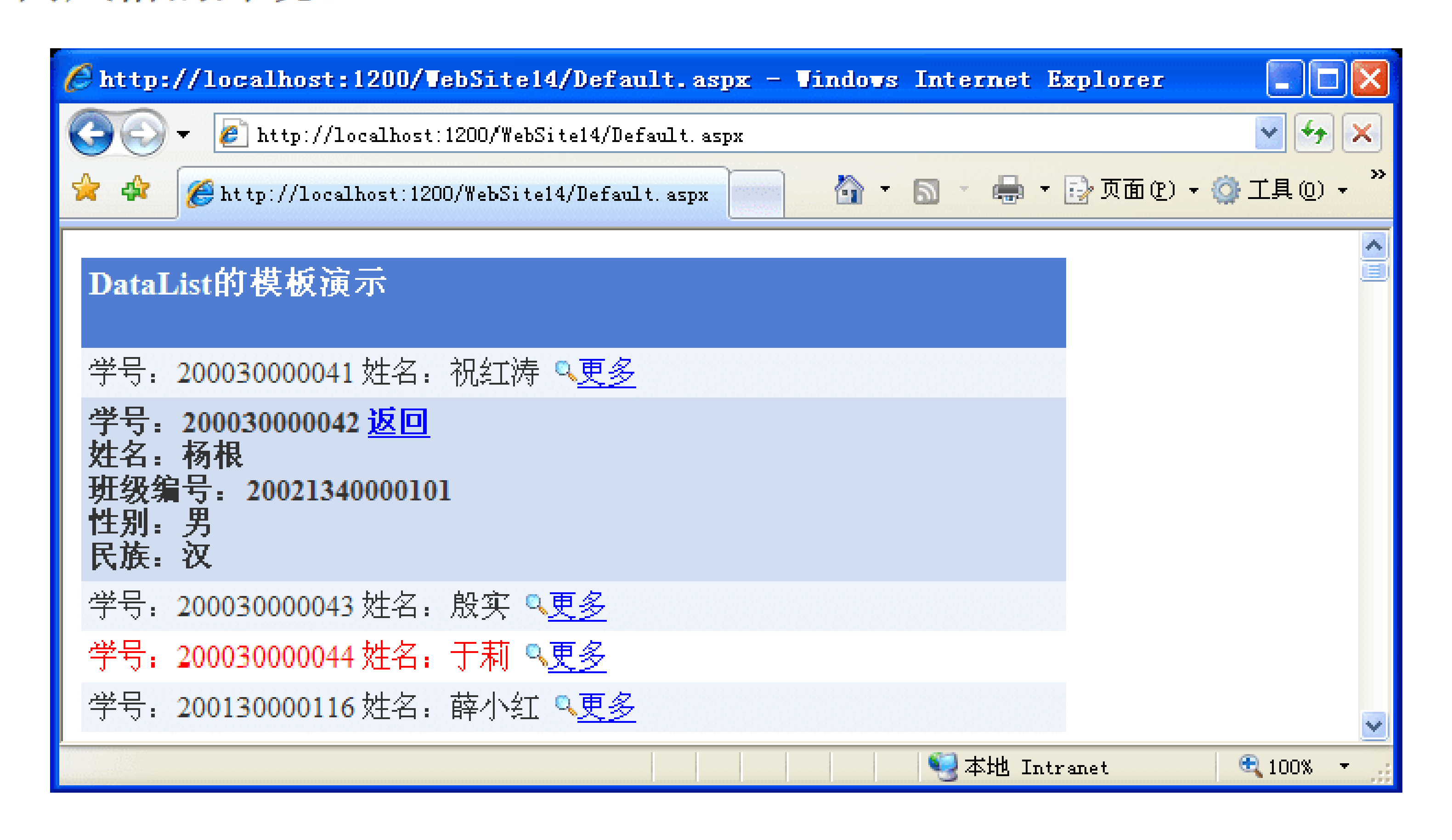


图 7-10　页面最终效果图

下面通过实例演示使用 GridView 控件显示、编辑和删除数据的方法，其中使用了上节的数据库源 SqlDataSource1。具体步骤如下所示。

（1）新建一个 ASP.NET 网页，使用上节介绍的方法将 SqlDataSource1 数据源控件添加到页面上。

（2）然后右击该控件选择【配置数据源】命令打开【配置数据源】对话框，在进入【配置 Select 语句】对话框时单击【高级】按钮。再从弹出的对话框中选择【生成 INSERT、UPDATE 和 DELETE 语句】复选框，并单击【确定】按钮返回，如图 7-11 所示。这样可以让数据源控件自动生成插入、更新和删除数据所需的 SQL 语句。

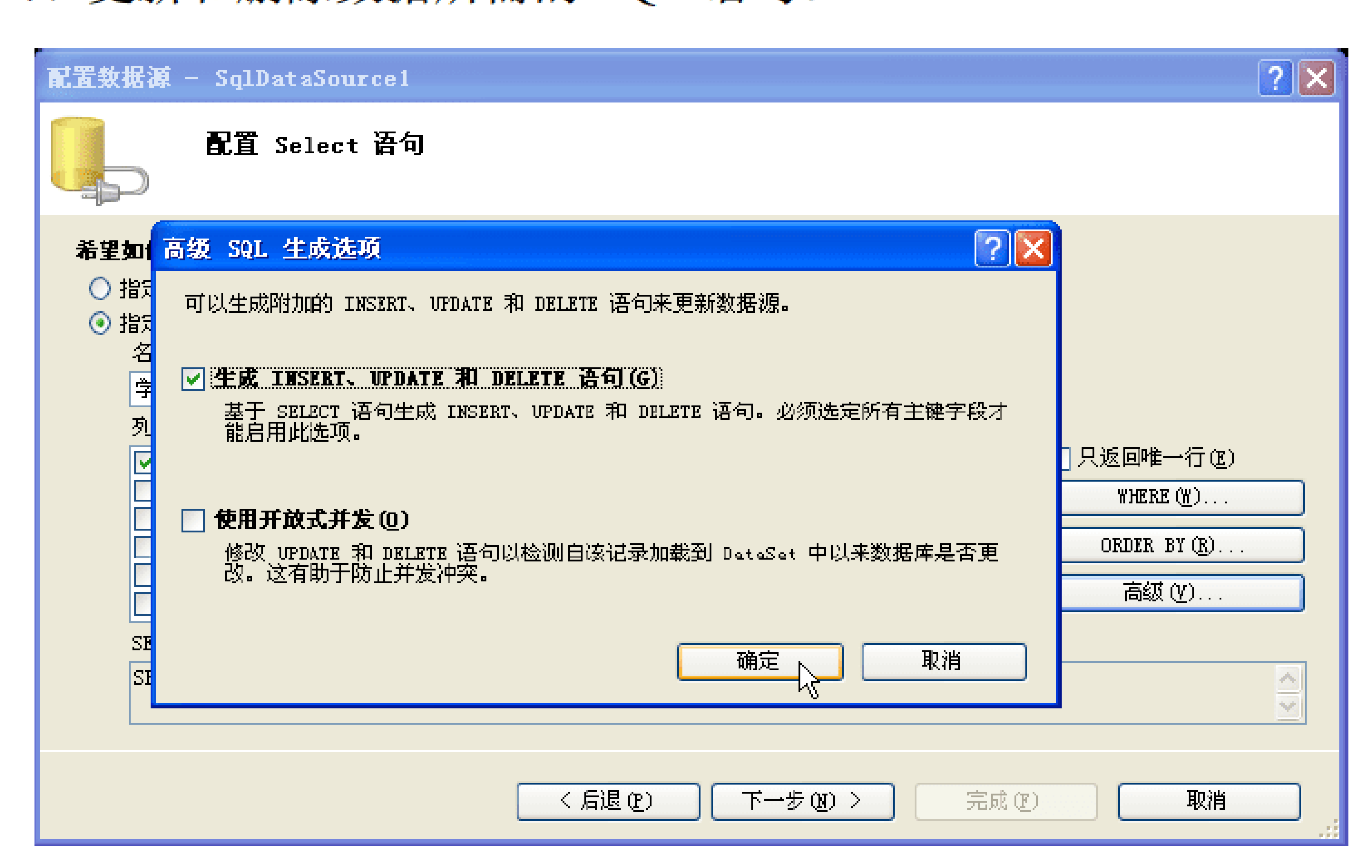


图 7-11　启用【生成 INSERT、UPDATE 和 DELETE 语句】复选框

（3）在页面上添加一个 GridView 控件。在【GridView 任务】面板中对控件进行设置，选择数据源为 SqlDataSource1，并选择【编辑】和【删除】复选框，如图 7-12 所示。

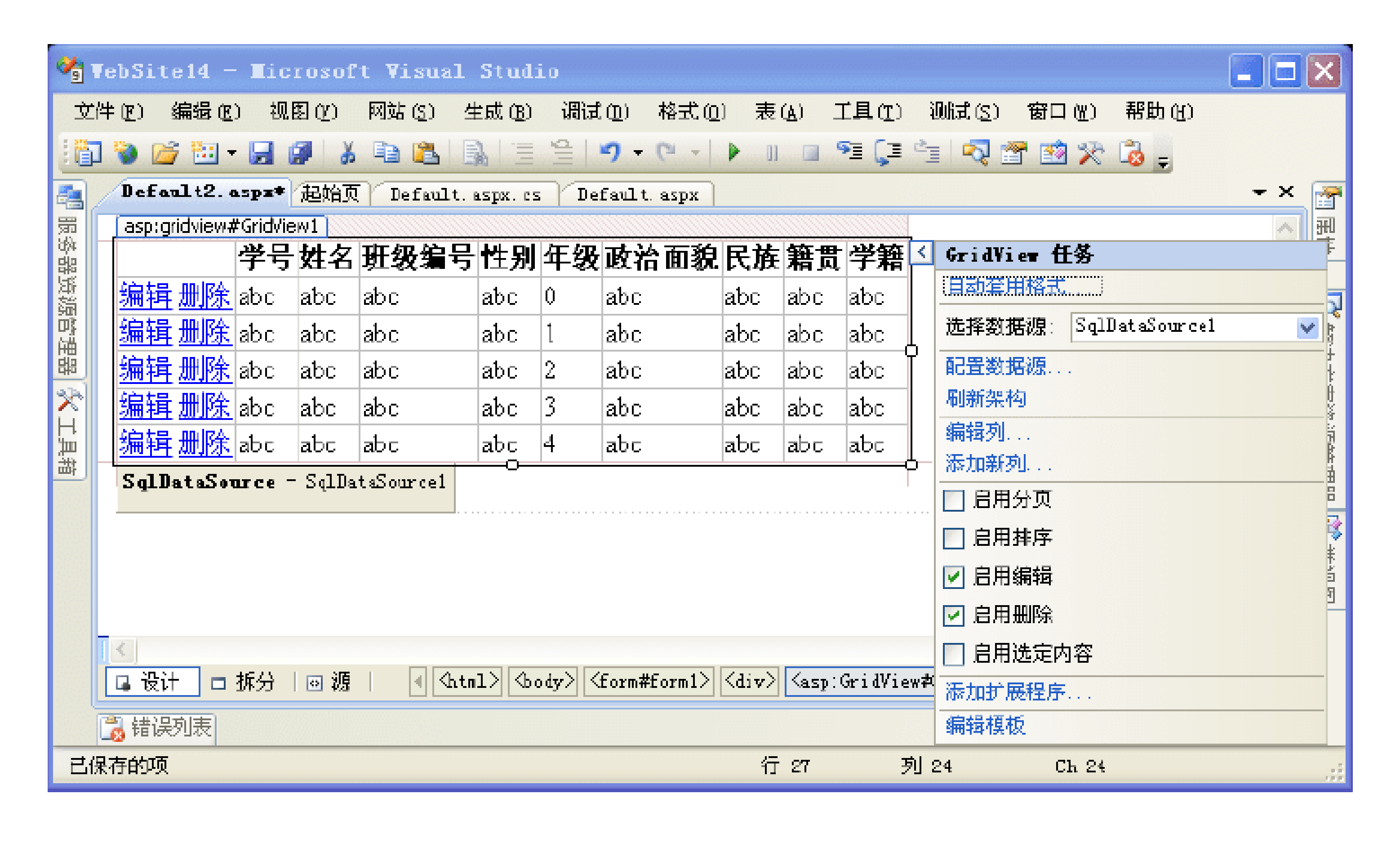


图 7-12　设置 GridView 控件属性

（4）单击【自动套用格式】按钮，在弹出的【自动套用格式】对话框中选择【传统型】架构，单击【确定】按钮返回。

（5）保存页面。右击选择【在浏览器中查看】命令，从浏览器中查看执行结果，如图 7-13 所示。使用套用的格式从数据库中显示了学生信息，并提供了【编辑】和【删除】链接。

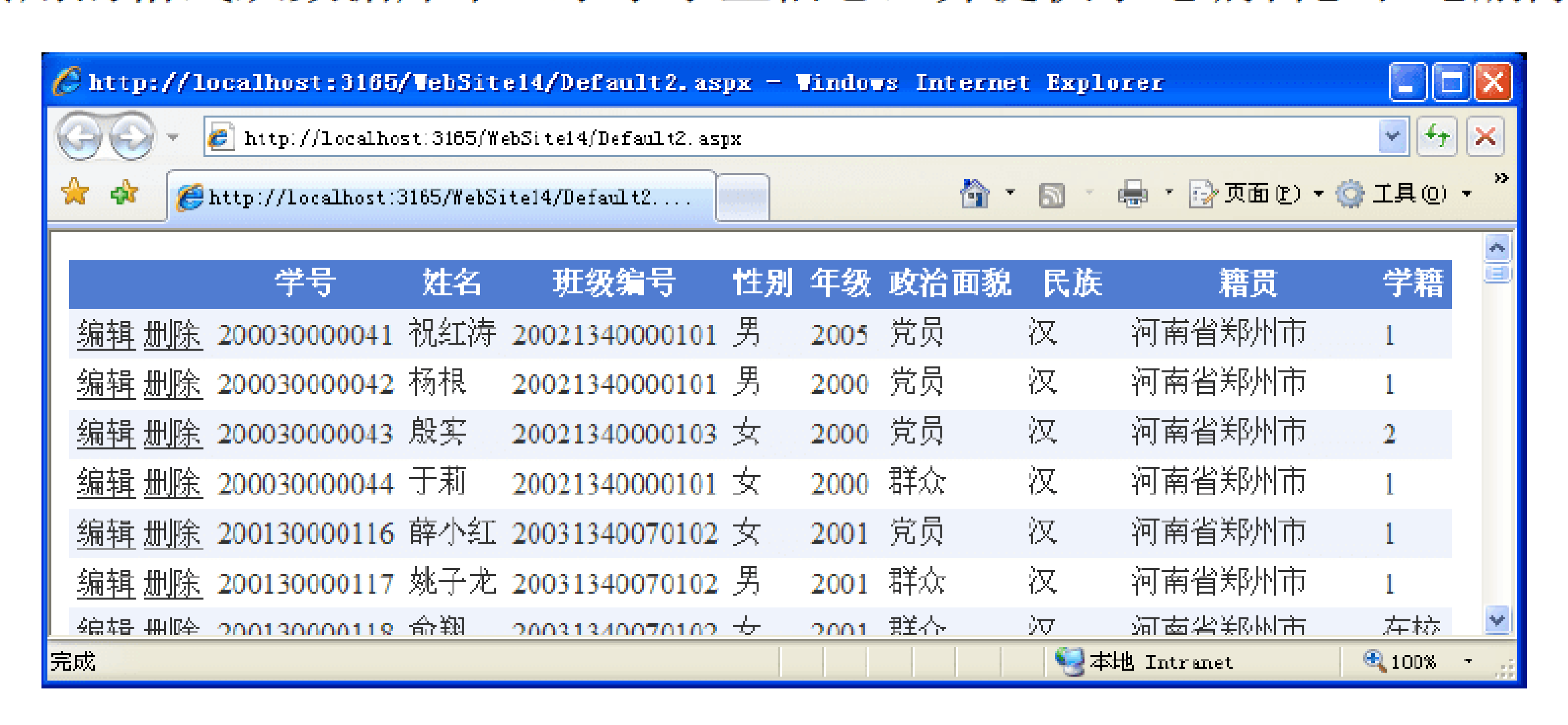


图 7-13　GridView 显示数据

（6）当用户需要对某行进行修改时，可以单击行前的【编辑】链接进入“编辑”状态，在各个文本框中输入新的值，如图 7-14 所示。然后单击【更新】链接保存，如果单击了【取消】链接将返回到原始状态。

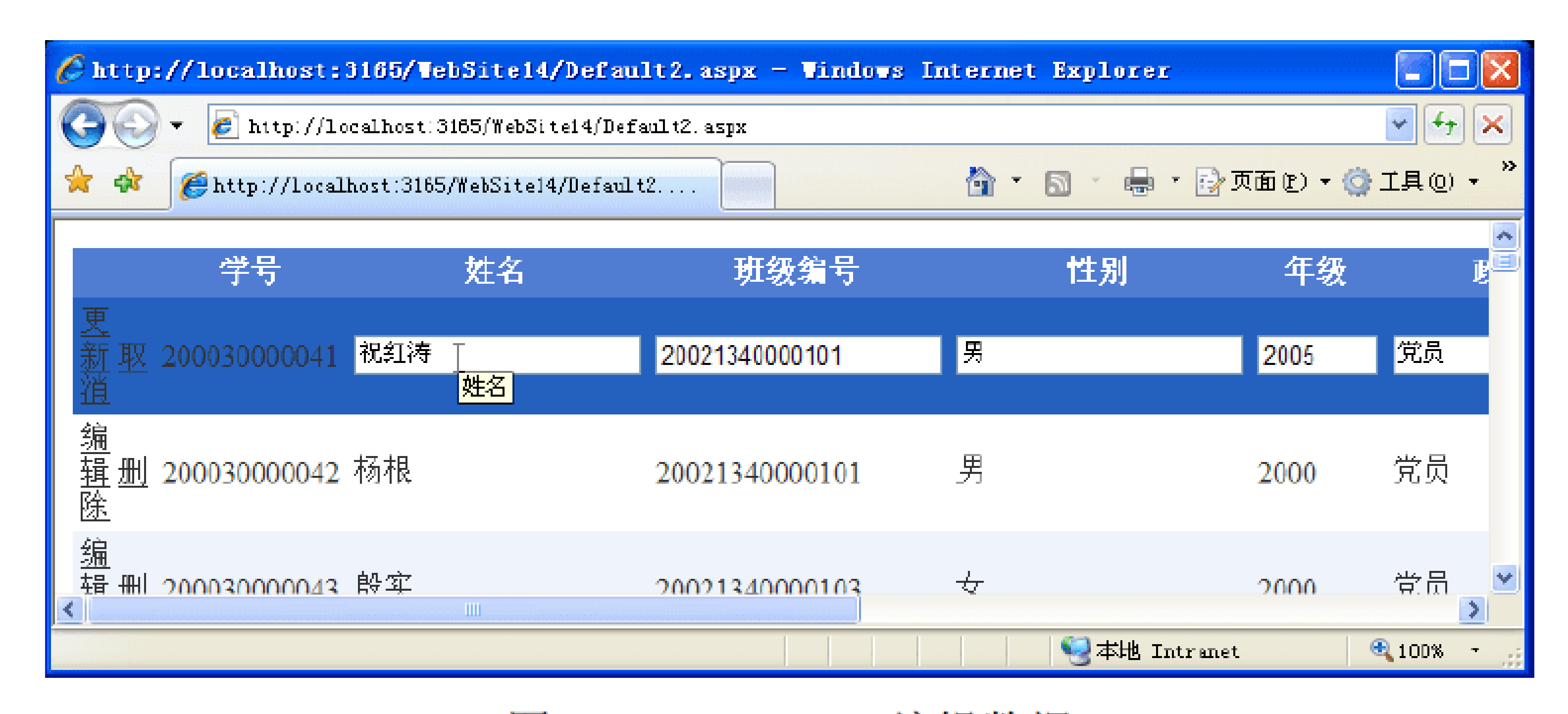


图 7-14　GridView 编辑数据

（7）最后，删除数据的操作是通过在行前单击【删除】链接完成的。

7.2.4　Repeater 控件

Repeater 控件没有默认的外观，因此可以使用该控件创建许多种列表，如表布局、逗号分割的列表和 XML 格式的列表等。当该页运行时，该控件为数据源中的每个项重复此布局。正因为 Repeater 控件没有默认的外观，所以 Repeater 控件显示数据时，必须先创建模板来绑定数据列表。模板可以包含标记和控件的任意组合。如果未定义模板，或者模板都不包含元素，则当应用程序运行时，该控件不显示在页面上。

Repeater 控件模板的定义一共有 5 种，分别如下所示。

- **ItemTemplate**

包含要为数据源中每个数据项都呈现一次的 HTML 元素和控件。该模板为数据源中的每一行都呈现一次的元素。若要显示 ItemTemplate 中的数据，必须声明一个或多个 Web 服务器控件并设置其数据绑定表达式以使其成为 Repeater 控件中的 DataSource 字段。在使用 Repeater 控件显示数据时，此模板为必选的。

- **AlternatingItemTemplate**

包含要为数据源中每个数据项都要呈现一次的 HTML 元素和控件。通常，可以使用此模板为交替项创建不同的外观，例如指定一种与在 ItemTemplate 中指定的颜色不同的背景色，即可以确定交替（从 0 开始的奇数索引）项的内容和布局。

- **HeaderTemplate**

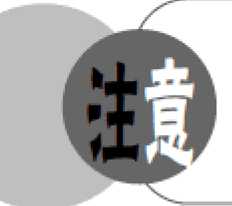
在所有数据绑定行呈现之前呈现一次的元素。典型的用途是开始一个容器元素，如表。如果在 Repeater 控件中定义了，则可以确定列表标头的内容和布局。如果没有定义该模板，则不呈现标头。

- **FooterTemplate**

在所有数据绑定行呈现之后呈现一次的元素。典型的用途是关闭在 HeaderTemplate 项中打开的元素，在使用 Repeater 控件时，如果定义了该模板，则可以确定列表注脚的内容和布局。如果没有定义，则不呈现注脚。

- **SeparatorTemplate**

包含在每项之间呈现的元素。典型的示例可能是一条直线（使用<hr/>标记），在使用 Repeater 控件时，如果定义了该模板，则呈现在项（以及交替项）之间。如果未定义，则不呈现分隔符。



该控件是不能可视化编辑模板的，并且 HeaderTemplate、FooterTemplate 和 SeparatorTemplate 都不能进行数据绑定。

下面创建一个实例，来具体说明该控件是如何使用模板绑定数据和显示数据的。该控件绑定数据和其他绑定控件是相同的，本实例主要说明该控件是如何布局、显示数据的。本实例具体开发步骤如下所示。

（1）打开前面 DataList 控件和 GridView 控件实例使用的网站，创建一个新的 ASP.NET

网页。

（2）重复前面介绍的方法，在页面上添加一个 SqlDataSource1 控件，并设置其数据库链接属性。

（3）在该页面上添加一个 Repeater 控件，然后切换到【源】视图为该控件添加 HeaderTemplate 模板。代码 7.13 所示为添加模板后 Repeater 控件的完整代码。

代码 7.13　添加 Repeater 控件和 HeaderTemplate 模板

```
<asp:Repeater ID="Repeater1" runat="server" DataSourceID=
"SqlDataSource1">
    <HeaderTemplate>
        <table width="100%" cellpadding="4" style="color: #333333">
            <tr style="background-color: #507CD1; font-weight: bold;
            color: #FFFFFF">
                <td colspan="9"><h3> 使用 Repeater 控件显示数据</h3></td>
            </tr>
            <tr><td> 学号</td> <td>姓名</td><td>班级编号</td><td>性别</td>
            <td>年级</td><td>政治面貌</td><td>民族</td><td>籍贯</td><td>学籍
            </td>
            </tr>
    </HeaderTemplate>
</asp:Repeater>
```

（4）添加一个显示项的 ItemTempLate，该模板用于显示数据，在该模板中我们使用 Eval() 方法绑定数据，该模板的主要代码如代码 7.14 所示。

代码 7.14　添加 ItemTempLate 模板

```
<ItemTemplate>
    <tr style="background-color: #EFF3FB">
        <td><%# DataBinder.Eval(Container.DataItem,"学号")%></td>
        <td><%# DataBinder.Eval(Container.DataItem,"姓名")%></td>
        <td><%# DataBinder.Eval(Container.DataItem,"班级编号")%></td>
        <td><%# DataBinder.Eval(Container.DataItem,"性别")%></td>
        <td><%# DataBinder.Eval(Container.DataItem,"年级")%></td>
        <td><%# DataBinder.Eval(Container.DataItem,"政治面貌")%></td>
        <td><%# DataBinder.Eval(Container.DataItem,"民族")%></td>
        <td><%# DataBinder.Eval(Container.DataItem,"籍贯")%></td>
        <td><%# DataBinder.Eval(Container.DataItem,"学籍")%></td>
    </tr>
</ItemTemplate>
```

（5）添加一个交替显示数据的模板 AlternatingItemTemplate，在该模板中编写如代码 7.15 所示的内容。

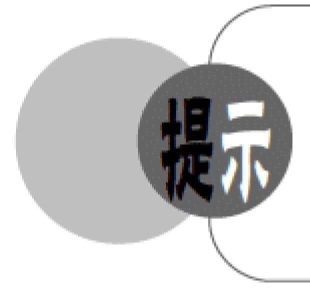
代码 7.15　添加 AlternatingItemTemplate 模板

```
<AlternatingItemTemplate>
   <tr style="background-color: #FFFFFF; color: #FF0000">
      <td><%# DataBinder.Eval(Container.DataItem,"学号")%></td>
      <td><%# DataBinder.Eval(Container.DataItem,"姓名")%></td>
      <td><%# DataBinder.Eval(Container.DataItem,"班级编号")%></td>
      <td><%# DataBinder.Eval(Container.DataItem,"性别")%></td>
      <td><%# DataBinder.Eval(Container.DataItem,"年级")%></td>
      <td><%# DataBinder.Eval(Container.DataItem,"政治面貌")%></td>
      <td><%# DataBinder.Eval(Container.DataItem,"民族")%></td>
      <td><%# DataBinder.Eval(Container.DataItem,"籍贯")%></td>
      <td><%# DataBinder.Eval(Container.DataItem,"学籍")%></td>
   </tr>
</AlternatingItemTemplate>
```

（6）最后添加一个用于结束数据的 FooterTemplate 模板，该模板在所有数据模板显示之后出现。这里把结束标记</table>放到 FooterTermPlate 模板中，如代码 7.16 所示。

代码 7.16　添加 FooterTermplate 模板

```
<FooterTemplate> </table></FooterTemplate>
```



AlternatingItemTemplate 模板和 ItemTemplate 模板中的数据显示时，是交替出现的，并且这两个模板的单元格设置的样式是不相同的。其中的 Eval()方法用在数据绑定表达式时，并根据浏览器的需要来格式化输出结果。

（7）最后单击 Repeater 控件右上角的智能按钮，选择在页面上添加的 SqlDataSource1 作为该控件的数据源。此时，该实例就成功地完成，设置该页面为起始页，运行程序，页面效果如图 7-15 所示。

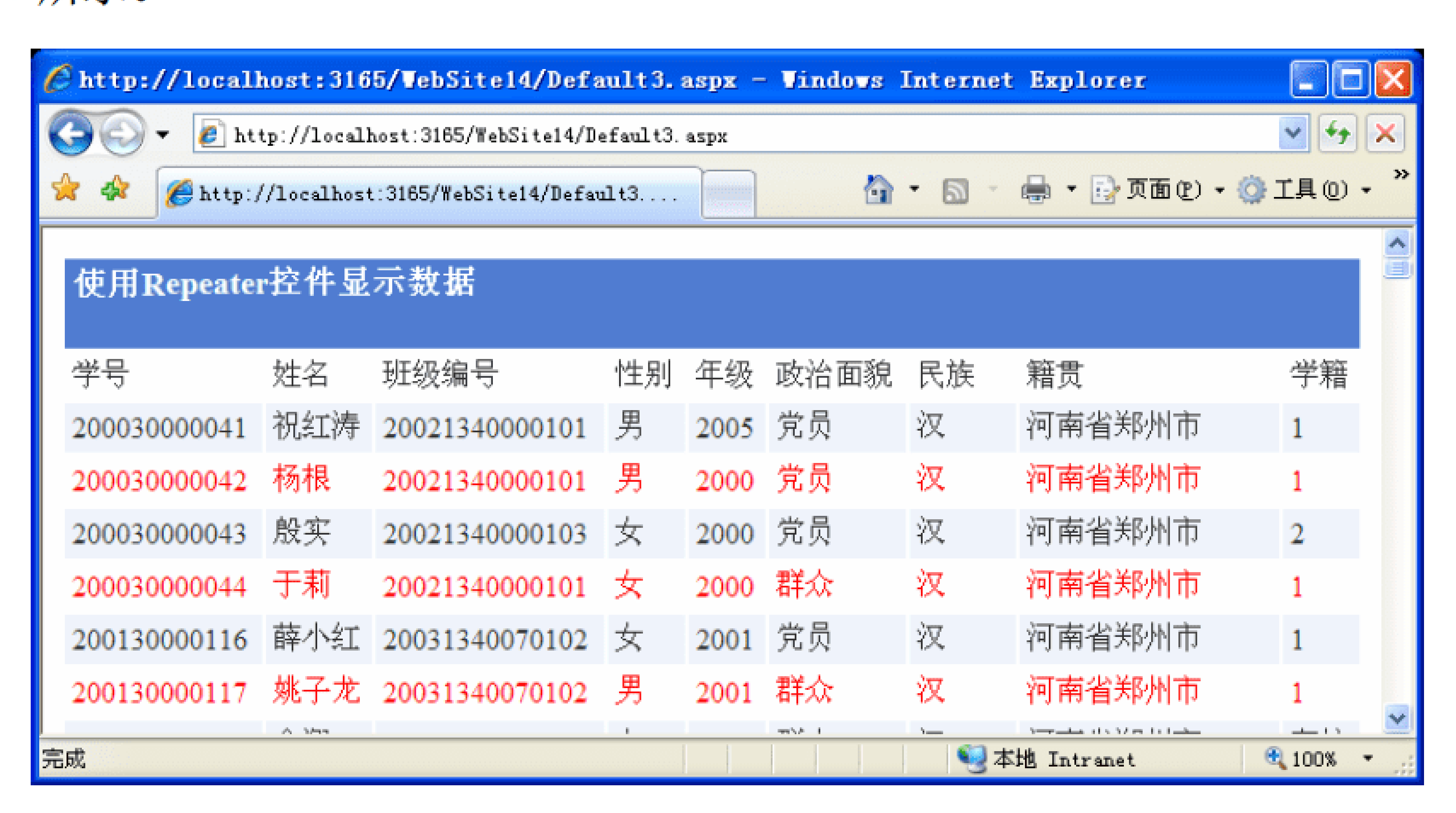


图 7-15　Repeater 控件显示数据

7.3 XML 命名空间和控件

XML 以其描述简单、功能强大、跨平台等特点被广泛应用在各种领域，逐渐成为一种新的网络数据处理方式。在网站中常被用来存储临时数据、配置信息或者列表详情等，使用 XML 有效减少了与数据库的交互次数，提升了服务器的访问性能和速度。

在 ASP.NET 中对 XML 的处理支持由 System.XML 命名空间中的类提供。使用它们可以方便、有效地完成获取 XML 元素、节点和值，定义 XML 文档结构和数据，不缓存数据顺序而遍历一个文件，格式转换、XML 与关系型数据的无缝集成和读写 XML 文件等功能。

System.xml 命名空间为处理 XML 提供了基于标准的支持，这些标准都由 W3C 发布，它们分别是下面几种。

- XML 1.0——http://www.w3.org/TR/1998/REC-xml-19980210-包括 DTD 支持。
- XML 命名空间——http://www.w3.org/TR/REC-xml-names/-流级别和 DOM。
- XSD 架构——http://www.w3.org/2001/XMLSchema。
- XPath 表达式——http://www.w3.org/TR/xpath。
- XSLT 转换——http://www.w3.org/TR/xslt。
- DOM 级别 1 核心——http://www.w3.org/TR/REC-DOM-Level-1/。
- DOM 级别 2 核心——http://www.w3.org/TR/DOM-Level-2/。
- SOAP 1.1——http://ww.w3.org/TR/SOAP。

ASP.NET 还为连接和访问 XML 数据提供了一个专用数据源控件——XmlDataSource。该控件用于连接和访问 XML 数据源中的数据。一旦 XmlDataSource 控件连接并获取了相关数据，就能够将这些数据与数据绑定控件绑定，并实现数据的有效显示。

XmlDataSource 控件的基本声明代码格式如下：

```
<asp:XmlDataSource ID="XmlDataSource1" runat="server"  DataFile=
"~/App_Data/Data.xml"
XPath="Node1/Node2">
</asp:XmlDataSource>
```

XmlDataSource 控件与其他控件一样，也具有属性和方法。表 7-1 所示为 XmlDataSource 控件的常用方法。

表 7-1　XmlDataSource 控件的方法

名称	说明
GetHierarchicalView()	用于获取 HierarchicalDataSourceView 对象，其参数 ViewPath 可以是一个有效的 XPath 值
GetXmlDocument()	将 XML 数据加载到内存中，这些数据来自数据源或者是缓存，并且返回相关的 XmlDocument 对象
OnTransforming(EventArgs e)	该方法用于定义事件 Transforming 的事件处理程序，相关内容参考 Transforming 事件的说明
Public void Save()	该方法用于将当前内存中的 XML 数据保存到由 DataFile 属性设置的 XML 文件中（如果 DataFile 属性已经设置）

XmlDataSource 控件常用属性详见表 7-2 所示。

表 7-2　XmlDataSource 控件的常用属性

属性	说明
CacheDuration	获取或者设置缓存数据在内存中存储的时间。单位为秒，默认值是无限大。当 EnableCaching 属性设置为 true，即启用数据缓存功能，这时应该将 CacheDuration 属性值设置为大于 0 的整数 *N*，在缓存作废之前，数据将在内存中保存 *N* 秒。另外，任何 Data 属性的改变或者 XML 文件内容的改变，都将引起缓存作废
DataFile	获取或设置一个值，该值指示 HTML 服务器控件是否显示在页面上
EnableCaching	获取或者设置一个布尔值，用于确定是否启用控件的数据缓存功能。值为 true，表示启用数据缓存功能，否则不启用此功能。默认值为 true
Transform	获取或设置一个 XSL 块，该块定义了对 Data 属性值或者 DataFile 属性所指定的 XML 数据进行 XSLT 的转换方式。默认值为空。如果同时设置了 Transform 和 TranformFile 属性，那么 TransformFile 属性将被优先使用。另外，如果设置了 XPath，那么该属性要等到 XML 文档转换后才被应用
TransformFile	获取或者设置 XSL 文件的路径，该 XSL 文件用于定义对 Data 属性值或者 DataFile 属性所指定的 XML 数据进行 XSLT 的转换方式。默认值为空
XPath	获取或者设置应用于 XML 数据中的 XPath 查询值。默认值为空

XmlDataSource 控件定义的事件只有一个 Transforming。该事件发生在由属性 Transform 或者 TransformFile 设置的 XSL 实现 XML 数据转换之前。可以定义 Transforming 的事件处理程序 OnTransforming。比较典型的应用是在 OnTransforming 处理程序中，设置 TransformArgumentList 属性以实现对 XSLT 参数列表的定义，该参数表用于由控件所加载的 XML 数据中。

7.4　显示 XML

在 ASP.NET 中可以使用 4 种方式来从一个 XML 文件中读取并显示信息，分别是：使用 XML 控件读取、使用 DOM 技术读取、使用 DataSet 对象和使用 XmlTextReader 类读取。下面依次对它们进行介绍，在开始之前首先介绍使用的 XML 文件 myCard.xml，内容如代码 7.17 所示。

代码 7.17　myCard.xml 文件内容

```
<?xml version="1.0" encoding="GB2312"?>
<?xml-stylesheet type="text/xsl" href="myCard.xsl"?>
<myCard>
   <User ID="E-081">
      <Name>王梦屹</Name>
      <Email>ayhncn@163.com</Email>
      <Career>销售人员</Career>
       <Site>http://www.itzcn.net</Site>
```

```
    </User>
    <User ID="E-214">
        <Name>祝红涛</Name>
          <Email>somboy@126.com</Email>
          <Career>软件开发工程师</Career>
          <Site>http://www.csdn.net</Site>
    </User>
    <User ID="E-070">
        <Name>贺强</Name>
          <Email>Qiange@188.com</Email>
          <Career>市场策划员</Career>
          <Site>http://www.webzcn.cn</Site>
    </User>
</myCard>
```

在上述 myCard.xml 文件中需要用到 myCard.xsl 对样式进行转换，它的内容如代码 7.18 所示。

代码 7.18　myCard.xsl 文件内容

```
<?xml version="1.0" encoding="GB2312"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
    <head>
    <title>我的通信录</title>
    </head>
    <body>   <xsl:apply-templates/>
    </body>
    </html>
  </xsl:template>
  <xsl:template match="User">
    <div style=" float:left; width:250px;"><b><xsl:value-of select="Name" />
    </b><br/>
      编号: <xsl:value-of select="@ID" /><br/>
      邮箱: <xsl:value-of select="Email" /><br/>
      职业: <xsl:value-of select="Career" /><br/>
      个人网站: <xsl:value-of select="Site" />
      <hr size="1" color="#000"/>
    </div>
  </xsl:template>
</xsl:stylesheet>
```

创建一个 ASP.NET 应用程序，将上述的两个文件放到网站的根目录中，然后再执行下面的操作。

7.4.1　XML 控件读取

使用 XML 控件读取 XML 文档是最简单的一种方法。使用这种方法时，只需要设置 XML 控件一个名为 DocumentSource 的属性，将值设置为要读取的 XML 文件的地址即可。例如，这里创建了一个名为 Default4.aspx 的文件，添加了 XML 控件并指定到 myCard.xml 文件。Default4.aspx 包含的内容如代码 7.19 所示。

代码 7.19　Default4.aspx 文件内容

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default4.aspx.cs"
Inherits="Default4" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http:
//www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>使用 XML 控件</title>
</head>
<body>
    <form id="form1" runat="server">
    <asp:Xml ID="Xml1" runat="server" DocumentSource="~/myCard.xml"
    TransformSource="~/myCard.xsl"/>
    </form>
</body>
</html>
```

保存并在浏览器中打开 Default4.aspx 即可看到 XML 控件通过 myCard.xsl 转换 myCard.xml 文件中内容后的效果，如图 7-16 所示。



图 7-16　使用 XML 控件读取 XML 文档

7.4.2　DOM 技术读取

.NET Framework 的 XML 类提供了一个符合 W3C DOM 标准的 XML 分析器对象 XmlDocument，它是在.NET 环境中执行大多数基于 XML 操作的核心对象。下面将举例说明

怎样使用 XmlDocument 对象读取 XML 文档。

在网站中新建一个 ASP.NET 页面，然后添加一个 XML 控件。注意，这里不需要设置控件的任何属性，此时页面代码如代码 7.20 所示。

代码 7.20　添加 XML 控件

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default6.aspx.cs"
Inherits="Default6" %>
    <asp:Xml ID="Xml1" runat="server"></asp:Xml>
```

打开后台.cs 文件，编写读取 XML 文件的代码，这里使用了 XmlDocument 对象。要先导入命名空间再编码，编写如代码 7.21 所示的内容。

代码 7.21　载入并显示 XML

```
protected void Page_Load(object sender, EventArgs e)  {
    XmlDocument doc = new XmlDocument();
    doc.Load(Server.MapPath("myCard.xml"));
    Xml1.TransformSource = "myCard.xsl";
    Xml1.Document = doc;
}
```

完成后运行这个页面，会看到与前面相同的效果。但是在该例中，同样使用了 XML 控件，但是并没有使用控件的 DocumentSource 属性，而是使用 XmlDocument 对象的 Load 方法载入了要读取的 XML 文档，然后将该 XmlDocument 对象与 XML 控件关联起来。

7.4.3　DataSet 对象读取

DataSet 将数据和架构作为 XML 文档形式进行读写。可以使用 WriteXmlSchema()方法将该架构保存为 XML 架构，并且使用 WriteXml()方法保存架构和数据。如果要读取既包含架构又包含数据的 XML 文档，可以使用 ReadXml()方法。

下面将举例说明怎样利用 DataSet 对象操作 XML 数据。在新建的 ASP.NET 页面中添加一个 GridView 控件，然后设置 ID 属性为 GridView1，再设置自动套用的格式。

在.cs 文件中编码代码读取 XML 数据到 GridView 控件并显示，如代码 7.22 所示。

代码 7.22　编写实现显示 XML 内容的代码

```
protected void Page_Load(object sender, EventArgs e)  {
    DataSet ds = new DataSet();
    ds.ReadXml(Server.MapPath("myCard.xml"));
    GridView1.DataSource = ds.Tables[0].DefaultView;
    GridView1.DataBind();
}
```

在该例中，XML 数据被加载到一个 DataSet 对象中，然后使用 DataGrid 控件显示出了它的内容。运行效果如图 7-17 所示，可以看出采用这种方法显示数据比前面两种清晰多了。



Name	Email	Career	Site	ID
王梦呓	ayhncn@163.com	销售人员	http://www.itzcn.net	E-081
祝红涛	somboy@126.com	软件开发工程师	http://www.csdn.net	E-214
贺强	Qiange@188.com	市场策划员	http://www.webzcn.cn	E-070

图 7-17　运行效果

7.4.4　XmlTextReader 类读取

利用 XmlTextWriter 类可以创建新的 XML 文件，而利用 XmlTextReader 对象可以读取磁盘文件，并以 XML 节点列表的形式显示数据。下面将举例说明如何以文本方式读取 XML 文档。

这种方法会将 XML 文档以文本形式读出，因此前台页面很简单，这里仅包含了一个名为 lter 的 Literal 控件。大部分代码是通过后台使用 XmlTextReader 类来完成的。例如，这里要实现读取 myCard.xml 文件，后台代码如代码 7.23 所示。

代码 7.23　XmlTextReader 类以文本方式读取 XML 文档

```
protected void Page_Load(object sender, EventArgs e) {
    //创建一个 XmlTextReader 实例，以读取 xml 文件
    XmlTextReader tr = new XmlTextReader(Server.MapPath("myCard.xml"));
    string strNodeResult = "";
    XmlNodeType nt;
    while (tr.Read())  {
        nt = tr.NodeType;
        switch (nt) {
            case XmlNodeType.XmlDeclaration:
                //读取 XML 文件头
                strNodeResult += "XML Declaration:<b>" + tr.Name
                + "" + tr.Value + "</b><br/>";
                break;
            case XmlNodeType.Element:
                //读取标签
                strNodeResult += "Element:<b>" + tr.Name + "</b><br/>";
                break;
            case XmlNodeType.Text:
                //读取值
                strNodeResult += " -Value:<b>" + tr.Value + "</b><br/>";
                break;
```

```
            }
            if (tr.AttributeCount > 0)   {
                while (tr.MoveToNextAttribute()){
                    strNodeResult += " -Attribute:<b>" + tr.Name
                    + "</b> Value:<b>" + tr.Value + "</b><br/>";
                }
            }
            lter.Text = strNodeResult;
        }
    }
```

在该例中，创建了一个 XmlTextReader 类的实例，然后使用该实例读取 XML 文件，显示的结果是以节点列表的形式输出的，如图 7-18 所示为运行效果。

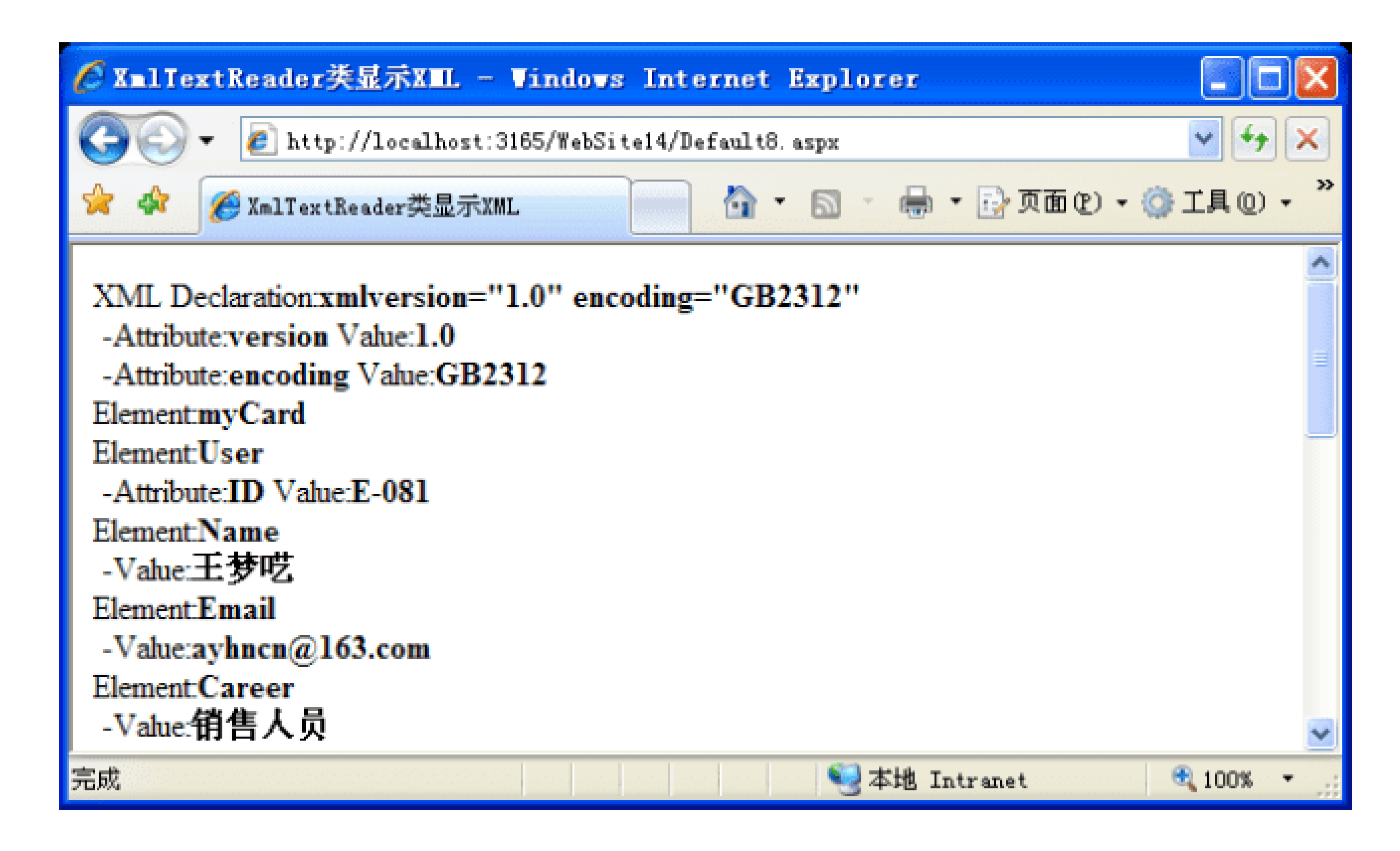


图 7-18　XmlTextReader 类读取 XML 文档

7.5　生成 XML

在 ASP.NET 中有很多种方法可以生成一个 XML 文档，其中最常用的是从数据库和文本中生成，本节会依次对它们进行介绍。

7.5.1　使用 DataSet 创建

使用 DataSet 对象的 WriteXml 方法可以方便地创建一个 XML 文件。由于 DataSet 属于 System.Data 命名空间，因此，很容易实现将一个数据库文件存储成一个 XML 文件。下面将举例说明如何使用 DataSet 创建 XML 文档。

接下来的操作需要用到数据库，因此首先导入所需的命名空间，如代码 7.24 所示。

代码 7.24　导入命名空间

```
using System.Data;
using System.Data.Sql;
```

```
using System.Data.SqlClient;
```

图 7-19 所示为用到的数据库，而且显示了 ClassList 表中的数据。接下来的工作，就是读取这个表中的内容，将它们保存到 XML 文件中。这样就不用在前台添加任何代码，代码 7.25 列出了后台实现代码。

代码 7.25　将 ClassList 表数据写入 XML

```
public SqlConnection conn;
public SqlDataReader dr;
public SqlDataAdapter da;
protected void Page_Load(object sender, EventArgs e)  {
string connStr = "server=ZHHT;uid=sa;pwd=123456;database=51uhui_SqlServer";
string sql = "select * from ClassList where parent=0";
conn = new SqlConnection(connStr);
conn.Open();
da = new SqlDataAdapter(sql, conn);
DataSet ds = new DataSet();
da.Fill(ds, "pdtSort");
ds.WriteXml(Server.MapPath("productSort.xml"));
}
```

程序执行完以后，就在当前目录下创建了一个名为 productSort.xml 的 XML 文件。如果使用浏览器打开该 XML 文件，结果如图 7-20 所示。在该例中，实现了将一个 SQL Server 2005 数据库文件转化为一个名为 productSort.xml 的 XML 文件。

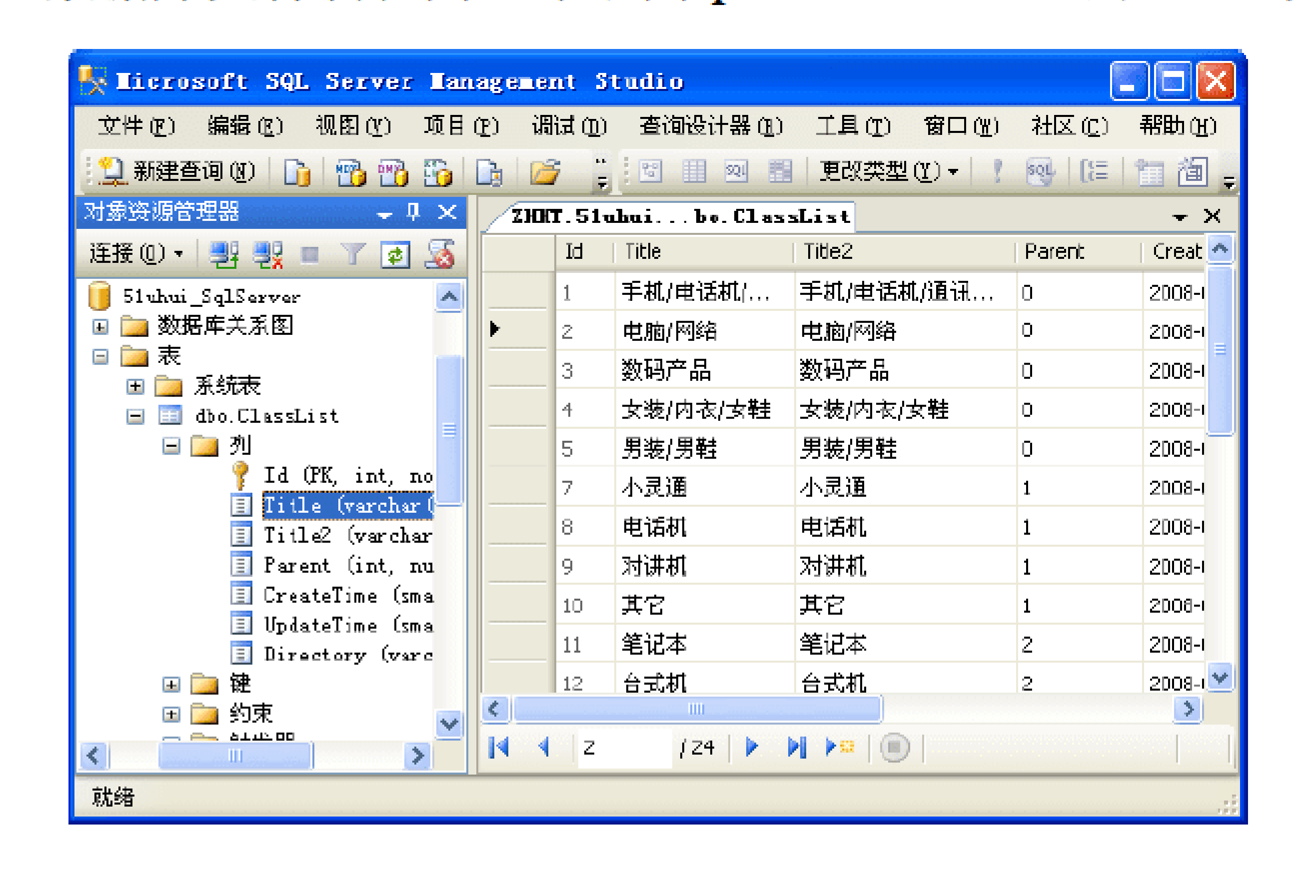


图 7-19　表数据

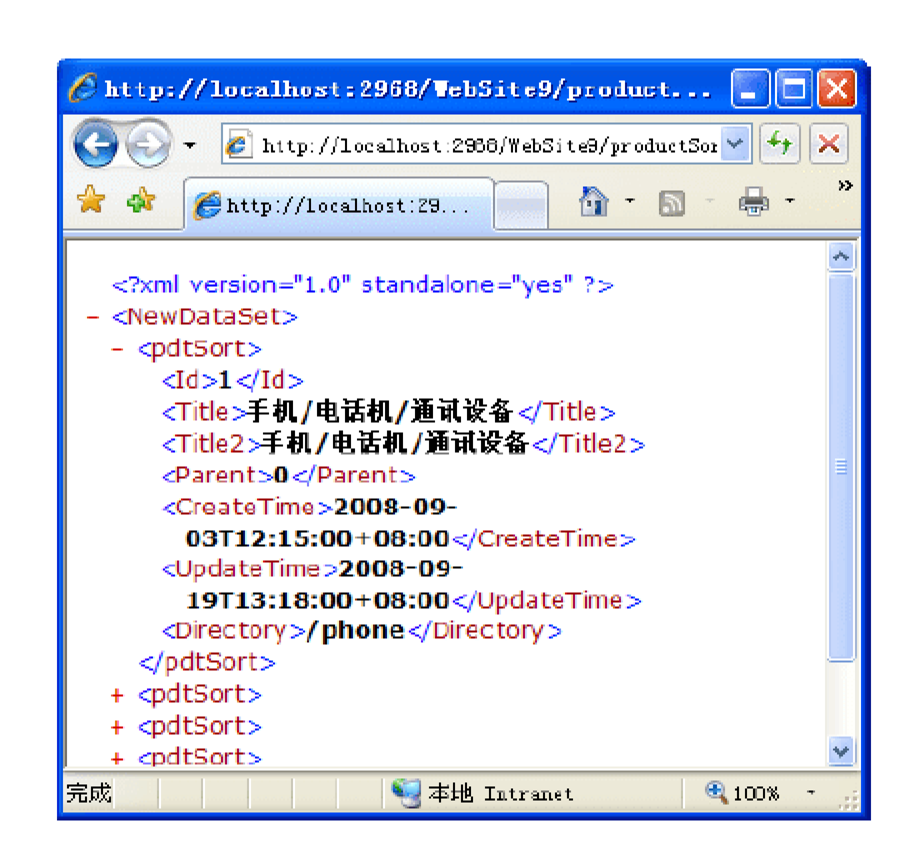


图 7-20　运行效果

7.5.2　使用文本方式创建

文本方式是指利用 XmlTextWriter 对象来创建 XML 文档，这个对象提供了很多方法可以方便容易地创建一个 XML 文档。下面将举例说明如何以文本方式创建 XML 文档。

XmlTextWriter 类位于 System.XML 命名空间中，它可以向 XML 文档中添加注释、属性、

节点以及子节点等。代码 7.26 所示为本例中创建 Children.xml 的代码。

代码 7.26　使用 XmlTextWriter 对象创建 XML 文档

```
protected void Page_Load(object sender, EventArgs e)
{
XmlTextWriter xmlTw = null;
xmlTw = new XmlTextWriter(Server.MapPath("Children.xml"), null);
xmlTw.Formatting = Formatting.Indented;
xmlTw.Indentation = 3;
xmlTw.WriteStartDocument();
xmlTw.WriteComment("已经使用 XMLTextWriter 创建完毕－" + DateTime.Now);
xmlTw.WriteStartElement("Books");
xmlTw.WriteStartElement("Book");
xmlTw.WriteAttributeString("Genre", "少儿科普读物");
xmlTw.WriteElementString("Cover_Art", "100000Why.jpg");
xmlTw.WriteElementString("Title", "十万个为什么");
xmlTw.WriteStartElement("Publisher");
xmlTw.WriteElementString("Name", "清华大学出版社");
xmlTw.WriteElementString("Date", "2008-8-25");
xmlTw.WriteEndElement();
xmlTw.WriteEndElement();
xmlTw.WriteEndDocument();
xmlTw.Flush();
xmlTw.Close();
}
```

运行这个程序后，会在网站的当前目录中创建一个名为 Children.xml 的 XML 文件，在浏览器中打开，会看到如图 7-21 所示的内容。

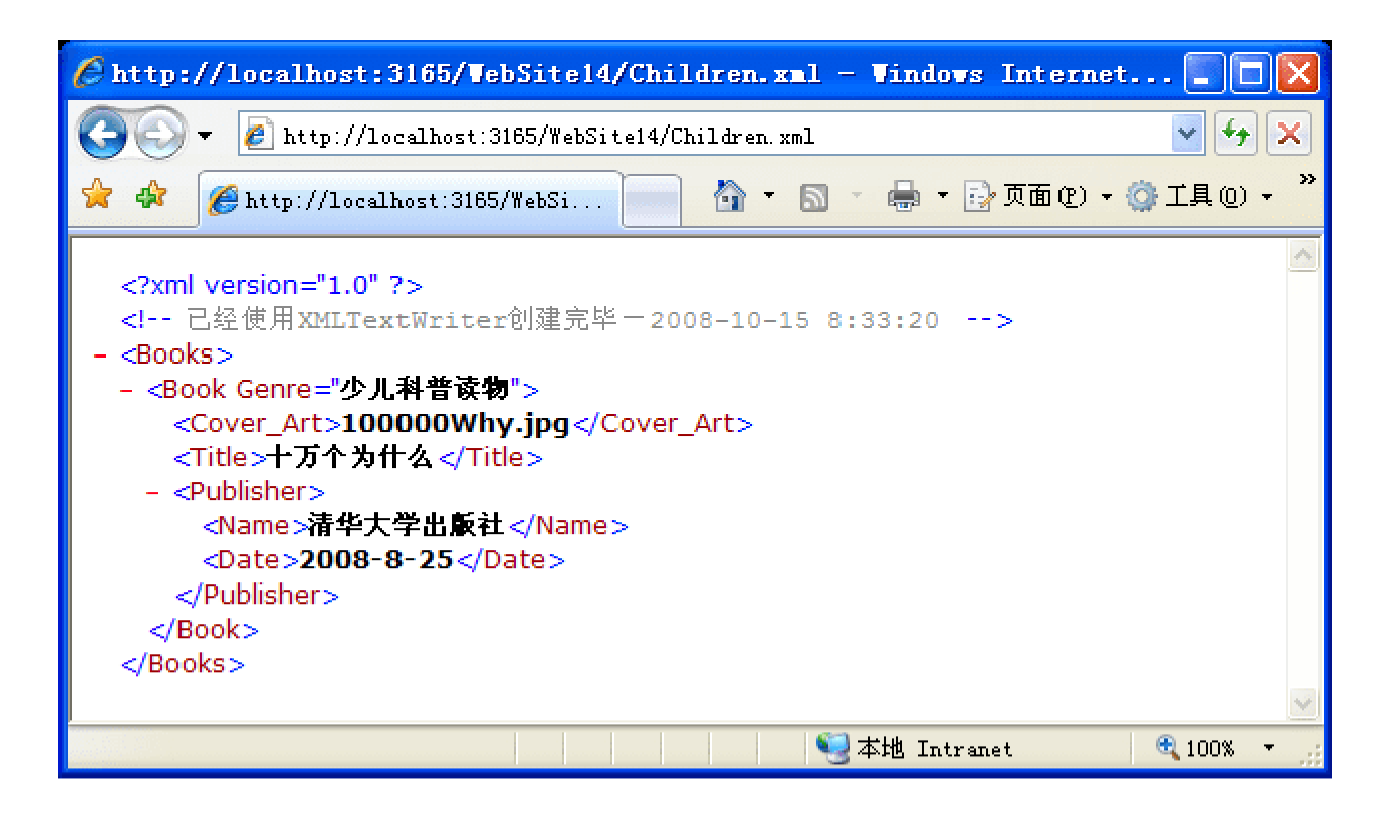
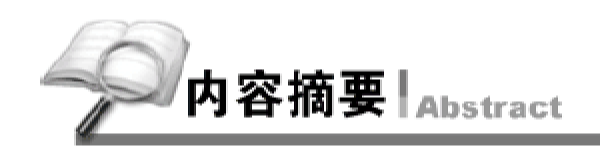


图 7-21　Children.xml 文件内容

第8章 ASP.NET 高级应用



内容摘要 | Abstract

ASP.NET 可以开发各种类型的网站，在网站开发中，一定会用到 ASP.NET 的内置对象。这些对象和 ASP 中的一样，只不过 ASP.NET 中功能更加强大、管理更加方便。ASP.NET 中的 Web 服务，能够使用户轻松创建和调用 Web 服务，实现代码共享。另外，ASP.NET 还提供文件处理功能，可实现轻松管理文件。本章将对 ASP.NET 的内置对象、Web 服务和处理文件进行逐一介绍。

学习目标 | Objective

- 掌握 Response、Request、Session 及 Cookie 对象的使用方法
- 熟悉 Server 及 Application 对象的使用方法
- 熟悉创建 Web 服务的过程
- 能够编写简单的 Web 服务
- 熟悉使用 Web 服务的过程
- 熟悉操作驱动器及文件夹的常用方法
- 掌握操作文件的常用方法

8.1 ASP.NET 内置对象

ASP.NET 内置对象是由 IIS 控制台初始化的 ActiveX DLL 组件。因为 IIS 可以初始化这些内置组件用于 ASP.NET 中，所以用户也可以直接引用这些组件来实现编程，即可以在应用程序中，通过引用这些组件来实现访问 ASP.NET 内置对象的功能。

ASP.NET 提供的内置对象有 Request、Response、Application、Session、Server 和 Cookies。这些对象使用户更容易收集通过浏览器请求发送的信息、响应浏览器以及存储用户的信息，以实现其他特定的状态管理和页面信息传递。在本章中将对这些内置对象和配置文件进行逐一介绍。

8.1.1 Response 对象

Response 对象用来访问所创建的客户端的响应，输出信息到客户端，它提供了标识服务

器和性能的 HTTP 变量，发送给浏览器的信息和在 Cookie 中存储的信息。它也提供了一系列用于创建输出页面的方法，例如经常会用到的 Response.Write 方法。该内置对象所属的类是 HttpResponse 类。

利用 Response 对象的相关属性和方法可以在 Page 页面上输出文字、转到网址、创建 Cookie 等。下面利用该对象的 Write 方法、Redirect 方法、添加 Cookie 的方法，实现不同的功能，说明 Response 对象的使用方法。代码 8.1 说明了 Write 方法的使用方式。

代码 8.1　Response 对象 Write 方法的使用

```
for (int i = 1; i < 7; i++)   {
    Response.Write("<p>");
    Response.Write("<font size='");
    Response.Write(i);
    Response.Write("'>");
    Response.Write(i.ToString()+"汇智科技");
    Response.Write("</font></p>");
}
```

上面的实例运用 Response 对象的 Write 方法，向客户端输出了由服务器端发送的消息。除了能在客户端输出消息外，Response 对象还能够将客户端浏览器重定向到另外的 URL 上，即跳转到指定的网页上。实现该功能只须使用 Response 对象的 Redirect 方法，该方法的使用方法如代码 8.2 所示。

代码 8.2　Redirect 方法的使用

```
string httpString = "http://www.webzcn.cn";
Response.Redirect(httpString);
```

ASP.NET 中包含两个内部 Cookie（Cookie 将在下面的章节中讲解到，此处就不再多做讲解）集合。通过 Response 对象可以对 Cookie 访问并且能为其添加一个新的 Cookie，代码 8.3 所示为 Response 对象添加 Cookie 的方法。

代码 8.3　添加 Cookie 的方法

```
HttpCookie myCookie = new HttpCookie("username");
DateTime now = DateTime.Now;
myCookie.Value = now.ToString();
myCookie.Expires = now.AddHours(1);
Response.Cookies.Add(myCookie);
```

8.1.2　Request 对象

Request 对象派生自 HttpRequest 类，该对象是用来获取客户端在请求一个页面或者传送一个 Form 时提供的所有信息，包括能够标识浏览器和用户的 HTTP 变量，存储在客户端的

Cookie 信息以及附在 URL 后面的值（指 URL 中“？”符号后面的查询字符串）查询字符串或者页面中<Form>段中的 HTML 控件内的值、Cookie、客户端证书、查询字符串等。可以使用此类读取浏览器已经发送的内容。

利用 Request 对象可以获得大量客户端信息，如可以获得客户端 IP、计算机 DNS 名称、获取当前请求的 URL 等值。这些都是通过 Request 对象的属性和方法获得的。下面通过代码演示 Request 对象的应用。

首先来看在程序中，经常使用 Request 对象的 QueryString 属性，来获得从上一个页面传递过来的字符串参数。例如在页面 1 中创建一个链接，使其指向页面 2，并用 QueryString 属性获得这两个变量，该连接代码如下所示。

```
<a href="74.aspx?username=myname&password=mypassword">查看详细</a>
```

上面创建好了链接，并传递两个值，一个是用户的 ID，另一个为用户名。现在在页面 2 中获得从页面 1 传递过来的两个值，具体操作是：在页面 2 的 Page_Load 事件中编写如代码 8.4 所示的内容。

代码 8.4　获得从页面 1 传递过来的字符串参数：74.aspx

```
protected void Page_Load(object sender, EventArgs e) {
    Response.Write("获得的用户名是:"+Request.QueryString["username"].ToString
    () + "<br/>");
    Response.Write("获得的密码是: " + Request.QueryString["password"].To
    String());
}
```

在该项目中右击 page1，从弹出的快捷菜单中选择【设为起始项】命令。然后单击工具栏中的【启动调试】按钮，运行程序。在页面 1 中单击【查看详细】链接跳转到页面 2，在页面 2 上显示的结果如下：

```
获得的用户名是: myname
获得的密码是: mypassword
```

在上面的例子中成功地获得了 QueryString 的值，利用 Request 对象还可以获得客户端的相关信息，如 IP、DNS 名称等。在代码 8.5 中就获得了客户端 IP 地址、DNS 名称、是否为本地计算机和请求的原始 URL，同样这些代码也编写在页面 2 的 Page_Load 事件处理程序中。

代码 8.5　获得客户端 IP 地址、DNS 名称等信息

```
Response.Write("客户端用户主机 IP: "+Request.UserHostAddress.ToString()+
"<br/>");
Response.Write("获取当前请求的原始 URL:" + Request.RawUrl.ToString() +
"<br/>");
Response.Write("获取远程客户端的 DNS 名称: " + Request.UserHostName + "<br/>");
Response.Write("是否为本地计算机: " + Request.IsLocal.ToString() + "<br/>");
```

运行代码 8.5，结果显示如下：

```
客户端用户主机 IP: 127.0.0.1
获取当前请求的原始 URL:/7-1/74.aspx?username=myname&password=mypassword
获取远程客户端的 DNS 名称: 127.0.0.1
是否为本地计算机: True
```

通过分析以上案例程序，了解了 Request 对象属性值的运用，下面利用 Request 对象提供的方法获取文件的物理路径，具体如代码 8.6 所示。

代码 8.6　获取文件的物理路径

```
string fileName = "75.aspx";
Response.Write("获得完整的物理路径: "+Request.MapPath(fileName));
```

该实例的程序运行结果如下所示。

```
获得完成的物理路径: F:\董红伟\书稿\Flex\源代码\7\7-1\75.aspx
```

8.1.3　Server 对象

Server 对象提供对服务器上方法和属性的访问，获取有关的最新错误信息，对 HTML 文本进行编码和解码。其中大多数方法和属性作为实用程序的功能服务。Server 对象也是 Page 对象的成员之一，主要提供一些处理页面请求时所需的功能，如建立 COM 对象、字符串的编、译码等工作。

Server 对象是类 HttpServerUtility 的一个实例，该类为 Server 对象提供了很多方法和属性，利用这些方法和属性能获取最新的错误信息以及对 HTML 文本进行编码和解码。下面通过实例代码说明 Server 对象的应用。

首先举一个返回服务器计算机名称和设定客户端超时期限的实例，该实例主要用到 Server 对象的 MachineName 属性和 ScriptTimeout 属性。打开 Visual Studio 2008 创建一个网站，在该网站.cs 文件中的 PAGE_Load 事件处理程序中，添加如代码 8.7 所示的内容。

代码 8.7　返回服务器计算机名称和设定客户端超时期限

```
Response.Write("服务器主机名为: " + Server.MachineName + "<br/>");
Server.ScriptTimeout = 10;
Response.Write("客户端超时期限为: " + Server.ScriptTimeout.ToString() +
"<br/>");
```

代码 8.7 的第一行代码获得了服务器主机名，第二行代码将客户端请求超时期限设置为 10 秒，如果在 10 秒内用户没有任何操作，服务器将断开与客户端的连接。该实例的运行结果如下所示。

```
服务器主机名为: ZZG
客户端超时期限为: 10
```

在上面的实例中对 Server 对象的属性进行了运用，下面利用 Server 对象的 HtmlEncode()

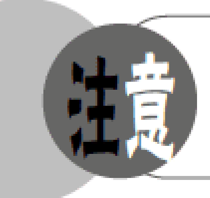
和 HtmlDecode()方法创建一个实例。代码 8.8 展示如何使用 HttpServerUtility 类的 HtmlEncode()方法和 HtmlDecode()方法，具体代码如下所示。

代码 8.8　HtmlEncode()方法和 HtmlDecode()方法的使用

```
string htmlString = "<a href=http://www.webzcn.cn>http://www.webzcn.cn</a>";
Response.Write("HtmlEncode 方法的结果: " + Server.HtmlEncode(htmlString)+"<br/>");
Response.Write("HtmlDecode 方法的结果: " + Server.HtmlDecode(htmlString) + "<br/>");
```

运行程序，读者会发现在页面上输出了两行文字，第一行输出的是“http://www.webzcn.cn”，而第二行输出的则是“http://www.webzcn.cn”的超链接。此时可以很清楚地看出这两个方法的区别，一个是对输出的内容不进行编码直接输出，而另一个 HtmlDecode()方法是对文字进行编码后输出。

使用 Server 对象的 MapPath()方法，可以将指定的相对或者虚拟路径，映射到服务器相应的物理目录上。该方法需要传递的参数是 Web 服务器的虚拟路径的字符串。若该字符串中以一个正斜杠（/）或者反斜杠（\）开始，则 MapPath()方法返回路径时将该字符串视为完整的虚拟路径。若 Path 不是以斜杠开始，则 MapPath()方法返回同页面文件中已有路径的相对路径。



MapPath()方法不检查返回的路径是否正确或者在服务器上是否存在。

8.1.4　Application 对象和 Session 对象

前面的章节介绍了 ASP.NET 提供的访问一个客户请求和产生响应的方法，本节介绍另外两个对象，Application 和 Session 对象。这两个对象不直接参与请求和响应的管理，而是在 Web 服务端对用户会话、服务器状态进行处理。

1. Application 对象

在 ASP.NET 环境下，Application 对象是 HttpApplictionState 类的一个实例。Application 对象，可以提供对所有会话应用程序范围内的方法和事件的访问，还提供对可用于存储信息应用程序范围缓存的访问。

下面通过实例的方式具体说明 Application 对象的创建，其属性和方法的运用。代码 8.9 中创建一个关于 Application 对象的实例，具体代码如下所示。

代码 8.9　创建 Application 对象

```
protected void Page_Load(object sender, EventArgs e){
    Application["username"] = "董红伟";
    Application.Add("sex", "男");
```

```
        Application.Add("age",26);
        for (int i = 0; i < Application.Count; i++)    {
            Response.Write(Application[i] + "  ");
            Response.Write(Application.Get(i) + "  ");
            Response.Write(Application.GetKey(i) + "<br/>");    }
       Application.Clear();
    }
```


在本例中首先为应用程序变量 Name0 设置了一个值，而后利用该对象的方法添加了两个 Application 对象，并赋予初值，接下来使用 Application 对象的 Count 属性获得该对象的总数（Application 对象是一个集合），然后通过循环操作该对象。该对象的 Get()方法，可以通过 Application 对象索引和名称获得该对象的值，而 GetKey()只能通过该对象的索引获得其名称，最后使用该对象的 Clear()方法清除 Application 对象集合。运行程序，页面显示结果如下：

```
董红伟 董红伟 username
男 男 sex
26 26 age
```

Lock()方法可以阻止其他客户修改存储在 Application 对象中的变量，以确保在同一时刻仅有一个客户可修改和存取 Application 对象中的变量。如果用户没有明确调用 Unlock()方法，则服务器将在页面文件结束或者超时后解除对 Application 对象的锁定。

Unlock()方法可以使其他客户端在使用 Lock 方法锁住 Application 对象后，修改存储在该对象中的变量。如果未显式地调用该方法，Web 服务器将在页面文件结束或者超时后解锁 Application 对象。使用方法如下所示：

```
Application.Lock();
Application["变量名"] = "变量值";
Application.UnLock();
```

2. Session 对象

Session 对象为当前用户会话提供信息，还提供对可用于存储信息会话范围缓存的访问，以及控制如何管理会话的方法。

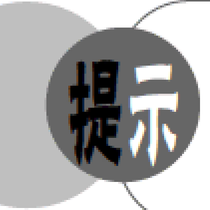
可以使用 Session 对象存储特定用户会话所需的信息。这样，当用户在应用程序的 Web 页面之间跳转时，存储在 Session 对象中的变量将不会丢失，而是在整个用户会话中一直存在下去。ASP.NET 的 Sessions 非常好用，能够利用 Session 对象来对 Session 全面控制，如果需要在一个用户的 Session 中存储信息，只需要直接简单地调用 Session 对象。

下面通过实例代码，说明 Session 对象的应用。首先创建一个网站应用程序，在【解决方案资源管理器】中双击 Default.aspx.cs 文件，打开该文件在 Page_Load 事件处理程序中编写如代码 8.10 所示的内容。

代码 8.10　保存数据到 Session 对象

```
protected void Page_Load(object sender, EventArgs e){
```

```
        if (!this.Page.IsPostBack)    {
            string username = "myname";
            string password = "mypassword";
            Session["username"] = username;
            Session["password"] = password;
            Session.Timeout = 1;
        }
    }
```



上面代码中的!this.Page.IsPostBack 表示的是，当页面第一次打开的时候值为 true，当页面回传时值为 false。IsPostBack 表示是否为页面回传。

在上一步骤中保存了 Session 值，并且设置 Session 值的过时时间为 1 分钟。下面在页面上添加一个服务器控件 Button 按钮，双击该按钮进入该按钮的 Click 事件处理程序中，在此处添加如代码 8.11 所示的内容。

代码 8.11　显示 Session 对象中的值

```
Response.Write("Session 保存的用户名的值: " + Session["username"]+"<br/>");
Response.Write("Session 保存的密码的值: " + Session["password"]);
```

至此该实例就成功地设计完成了，运行程序单击 Button 按钮，显示结果如下所示：

```
Session 保存的用户名的值: myname
Session 保存的密码的值: mypassword
```

过 1 分钟后再单击该按钮，显示的结果为：

```
Session 保存的用户名的值:
Session 保存的用户名的值:
```

从结果中可以看出，两次的值不同，这就说明保存的 Session 值已经过时，Session 已经不再保存该值。本实例讲解了如何保存和读取 Session 值，当然也可为 Session 对象添加值，这就要用到 Session 对象的 Add()方法。

8.1.5　Cookie 对象

Cookie 是一小块由浏览器存储在客户端系统上的文本，是一种标记。由 Web 服务器嵌入用户浏览器中，以便标识用户，且随同每次用户请求发往 Web 服务器。用户访问 Web 站点时，每个站点都可能会向用户浏览器发送一个 Cookie，而浏览器会将所有这些 Cookie 分别保存。Cookie 是 Web 站点而不是具体的页面，所以无论用户请求浏览站点中的那个页面，浏览器和服务器都将交换 Cookie 信息。

Cookie 跟 Session、Application 类似，也是用来保存相关信息，但 Cookie 和其他对象的最大不同是，Cookie 将信息保存在客户端，而 Session 和 Application 是保存在服务器端。也就

是说，无论何时用户连接到服务器，Web 站点都可以访问 Cookie 信息。这样，既方便用户的使用，也方便了网站对用户的管理。

浏览器负责管理用户系统上的 Cookie。创建 Cookie 时，需要指定 Name 和 Value，还可以设置 Cookie 的到期日期和时间。用户访问编写 Cookie 站点时，浏览器将删除过期 Cookie。

如果没有设置 Cookie 的有效期，则不会将其存储在用户的硬盘上，而会将 Cookie 作为用户会话信息的一部分进行维护。当用户关闭浏览器时，Cookie 便会被丢弃。这种非永久性 Cookie 很适合用来保存只需短时间存储的信息，或者保存由于安全原因不应该写入客户端计算机磁盘的信息。已经了解了 Cookie，现在就在 ASP.NET 中创建一个 Cookie，具体创建方法如代码 8.12 所示。

代码 8.12　通过 HttpCookie 对象创建 Cookie

```
HttpCookie myCookie = new HttpCookie("username");
myCookie.Value = "MyName";
myCookie.Expires = DateTime.Now.AddDays(1);
Response.Cookies.Add(myCookie);
```

在代码 8.12 中创建了一个 Cookie，使用另一种方法同样也可以创建 Cookie，具体方法如代码 8.13 所示。

代码 8.13　创建 Cookie 的另一种方法

```
Response.Cookies["username"].Value = "MyName";
Response.Cookies["username"].Expires = DateTime.Now.AddDays(1);
```

这样就创建了一个 Cookie 对象，右击浏览器属性弹出【Internet 属性】对话框，单击【浏览历史记录】窗格下的【设置】按钮，弹出【Internet 临时文件和历史记录设置】对话框，单击【查看文件】按钮，会发现创建的 Cookie 文件。

在上面的例子中创建了一个 Cookie，如果用户设置为拒绝接受 Cookie，在不能写入 Cookie 时是不会引发任何错误的，同样浏览器也不会向服务器发送有关当前 Cookie 设置的任何信息。确定 Cookie 是否被接受的一种方法是尝试编写一个 Cookie，然后再尝试读取该 Cookie，如果无法读取该 Cookie，则可以确定浏览器不接受 Cookie。

下面以一个实例演示如何测试浏览器是否接受 Cookie。此实例由两个页面组成。第一个页面写出 Cookie，然后将浏览器重定向到第二个页面，第二个页面尝试读取该 Cookie。第一个页面的具体代码如代码 8.14 所示。

代码 8.14　写入 Cookie

```
protected void Page_Load(object sender, EventArgs e){
    if (!Page.IsPostBack)    {
        Response.Cookies["Mycookie"].Value = "CookieOK";
        Response.Cookies["Mycookie"].Expires = DateTime.Now.AddMinutes(1);
        Response.Redirect("711.aspx");
    }
}
```

该页面转到第二个页面后，在第二个页面中添加如代码 8.15 所示的内容。

代码 8.15　尝试读取 Cookie

```
protected void Page_Load(object sender, EventArgs e){
    string redirect = Request.QueryString["redirect"];
    if (Request.Cookies["Mycookie"] == null)    {
        Response.Write("浏览器不能接受 Cookie");
    }
    else    {
        Response.Write("浏览器可以接受 Cookie 值，该 Cookie 值为: " + Request.
        Cookies["Mycookie"].Value);
    }
}
```

8.2　Web 服务

Web 服务从理论上说并不是新概念，它允许分布式应用程序通过网络共享业务逻辑。Microsoft 的 Web 服务使用标准的 Web 协议（如 HTTP）和数据描述语言（如 XML），通过公用商品交换数据，并利用了无所不在的 HTTP 基础结构支持。Web 服务从技术上来说是通过标准的 Web 协议访问的可编程的应用程序逻辑。这个定义包括可编程的应用程序逻辑和标准 Web 协议。

8.2.1　Web 服务概述

目前访问 Internet 信息的最常用方法是通过浏览器发送和接收消息，这些消息包含使用 HTTP 协议传输的 HTML 标记，然后 HTML 标记在 Web 浏览器中分析，并显示用户界面。

Web 服务（Web Service）就是一种应用程序，可以使用标准的互联网协议（如 HTTP 等）将功能纲领性地体现在互联网和企业内部网上，可将 Web 服务视为 Web 上的组件编程。Web 服务的出现被认为是分布式计算领域的一项重大飞跃，预示着：当用户需要某种功能时，可以在 Internet 上以编程方式找到并访问。这种基于 HTTP 和 XML 的技术由于其非专用的实现、开放的协议和方便的跨平台性等诸多优点，成为目前分布式应用的主要方式。

1. 什么是 Web 服务

“Web 服务”是 Web 服务器所提供的一个应用程序或者可执行代码的程序块，其功能可以通过标准的 XML 协议展示。

从表面上看，Web 服务就是一个应用程序，它向外界公布了一个能够通过 Web 进行调用的 API。也就是说，能够用编程的方法通过 Web 来调用该程序。Web 服务平台是一套标准，定义应用程序应如何在 Web 上实现相互操作。用任何语言、在任何平台上写的 Web 服务程序

都可以通过 Web Service 标准对这些服务进行查询和访问。

Web 服务是一种全新的 Web 应用程序分支，是自包含、自描述、模块化的应用，可以发布、定位，也可以通过 Web 调用。Web 服务可以执行从简单请求到复杂商务处理的任何功能。一旦部署之后，其他 Web 服务应用程序可以发现并调用部署的服务。

Web 服务是一种应用程序，可以使用标准的互联网协议（如 HTTP 协议）将功能纲领性地体现在互联网和企业内部网上，可将 Web 服务视为 Web 上的组件编程。

从传统意义上来说，.NET Web 服务并不是一个对象。在本质上，Web 方法是非常独立、抽象和基本的，Web 服务更像 DLL 中的一个函数库，而不是真正面向对象的抽象，这种简化是 Web 服务一个重要优点。因为 Web 服务并没有紧密绑定到某种特定的安全机制、状态管理或者传输技术，所以可以用于任何开发方案。

2. Web 服务的作用

Web 服务是一种共享编程功能的方法，可以通过以下几种方式使用 Web 服务。

- **支持企业对企业的交易或者连接各个公司的内部系统**　这是目前一段时间内使用 Web 服务的常见方法。Web 服务可以允许文档和知识共享或者相关服务的集成。例如，Web 服务可以用来帮助电子商务公司自动与货运公司联系，填写订单。这时，Web 服务使用者可能是在组织内部使用的软件。
- **作为开发人员的预创建模块**　例如，第三方开发人员可以创建用于认证的 Web 服务，供 ASP.NET 站点使用。Web、台式机和可移动应用程序可以很容易地使用这种预先创建的组件。
- **作为客户应用程序的增值产品特性**　例如，Microsoft 公司希望 IT 部门部署 Windows 操作系统，那么该公司就有必要使用一种允许系统管理员使用 Web 服务执行远程管理的技术。
- **作为可重用的组件 DLL**　例如，在 ASP.NET 应用程序中重用某种功能的最容易的方法不是创建.NET 程序集，而是设计各种客户都可以使用的 Web 服务。无须担心 Web 服务和客户的位置，只需确保每一个客户都拥有 Internet 连接即可。
- **作为连接同一家公司中软件包的工具**　例如，Web 服务可以用来在一个安全的公司网络上将专用工资单软件连接到记账软件。

Web 服务并不是最终用户产品，而是一些基于组件的应用程序，允许在不同环境中和不同类型的客户上重用业务逻辑。Web 服务的使用者总是另一个应用程序。

3. .NET Web 服务

Web 服务的 Microsoft 实现方式用来使调用远程 Web 服务就像调用本地类上的一个方法那样容易。为了实现这一点，.NET Framework 提供了一些工具来隐藏 SOAP 和 WSDL 等标准的一些细节。该过程的工作方式如下所示，其中，第 1、3、5、8 步是手工执行的步骤。

（1）将一项 Web 服务设计为一个.NET 类，其具有一些将其标识为包含已表述函数的 Web 服务。

（2）.NET 自动创建一个 WSDL 文档，该文档说明了客户如何与 Web 服务进行通信。

（3）客户发现 Web 服务并使用。客户将其作为一个 Visual Studio.NET 项目的 Web 引用来

添加（或者运行 WSDL.exe 实用程序）。

（4）.NET 自动检查 WSDL 文档并生成一个允许客户透明地与 Web 服务进行通信的代理类。

（5）客户调用 Web 服务类的一个方法。在客户看来，这好像与调用其他任何类中的方法一样，但事实上，客户正在与代理类而不是 Web 服务进行交互。

（6）在幕后，代理类将提供的参数转换为 SOAP 消息并将其发送给 Web 服务。

（7）代理类在短时间内收到一个 SOAP 回复，将其转换为适当的数据类型，作为一种普通的.NET 数据类型返回给客户。

（8）客户使用返回的信息。

.NET Web 服务使用了 ASP.NET 技术，该技术是.NET Framework 的一部分，并需要 IIS 的服务技术。Visual Studio.NET 提供了许多工具和一个复杂的 IDE 来简化创建和运行 Web 服务所需的工作。

4. 基础技术

要创建或者使用 Web 服务，无须知道很多与基础技术有关的知识。但是，为了能够设计出能利用最佳平台特性和巧妙地避免常见错误的 Web 服务，了解基础技术知识是有益的。

Web 服务使用不同开放式标准的组合，如表 8-1 所示。

表 8-1　Web 服务技术

技术	说明
WSDL	一种基于 XML 的格式，描述了一项 Web 服务，列举其方法、所有参数和返回值所使用的数据类型以及通信的支持方法（或者绑定）
HTTP	用来在 Internet 上发送 Web 请求和响应的通信协议。也可用作在 Web 浏览器中检索 Web 页面的标准
SOAP	一种基于 XML 的格式，在 Internet 上发送 Web 服务请求和响应消息之前，将对这些消息进行编码。例如，SOAP 指定了应该如何表示不同数据类型的值
DISCO	允许客户查找 Web 服务的可选 Microsoft 规范。事实上，DISCO 文件就是一个未分类的 Web 服务链接列表，将被 WS-Inspection 标准所取代，但该标准还没有集成到.NET Framework 中
UDDI	一个目录，允许客户查找特定公司展示的 Web 服务。UDDI 是最新 Web 服务标准

其中，WSDL（Web Service Description Language，Web 服务描述语言）是唯一与.NET Web 服务紧密结合的标准。虽然建议使用 SOAP（Simple Object Access Protocol，简单对象访问协议，与其他 Web 服务开发平台兼容所必须的），但是也支持 HTTP POST 和 HTTP GET 传输。DISCO 和 UDDI（Universal Description Discovery and Integration，通用描述、发现和集成）都是可选的扩展，可以使发布和发现 Web 服务信息更加容易，但并不是必须的。

除了表 8-1 中列举的标准之外，有一些 Web 服务标准还没有进入.NET 领域，例如 WS-Inspection 用于发现文档的规范，列出了几组 Web 服务及其终结点。WS-Inspection 是由 Microsoft 和 IBM 共同开发的，将会代替 DISCO 成为.NET 平台使用的本机发现机制，但是目前.NET 版本仍然使用 DISCO 标准。

8.2.2 创建 Web 服务

使用.NET Framework 很容易创建和运行一个基本 Web 服务，而不需要了解任何 HTTP、SOAP、WSDL 或者任何形成 Web 服务的基础技术。本节将会介绍如何使用 Visual Studio. NET 创建和测试一些简单的 Web 服务。

打开 Visual Studio 2008，选择【文件】|【新建】|【网站】命令，然后选择“ASP.NET Web 服务”选项，打开如图 8-1 所示的对话框，在【位置】下拉列表框中选择文件系统，并通过【浏览】按钮得到网站创建地址，使用的【语言】选择 Visual C#，然后单击【确定】按钮。



图 8-1　创建 Web 服务对话框

这时，在创建网站的目录中，已经包含了一个 Web 服务文件，该文件名为 Service.asmx；解决方案资源管理器默认包含的文件夹和文件，如图 8-2 所示。

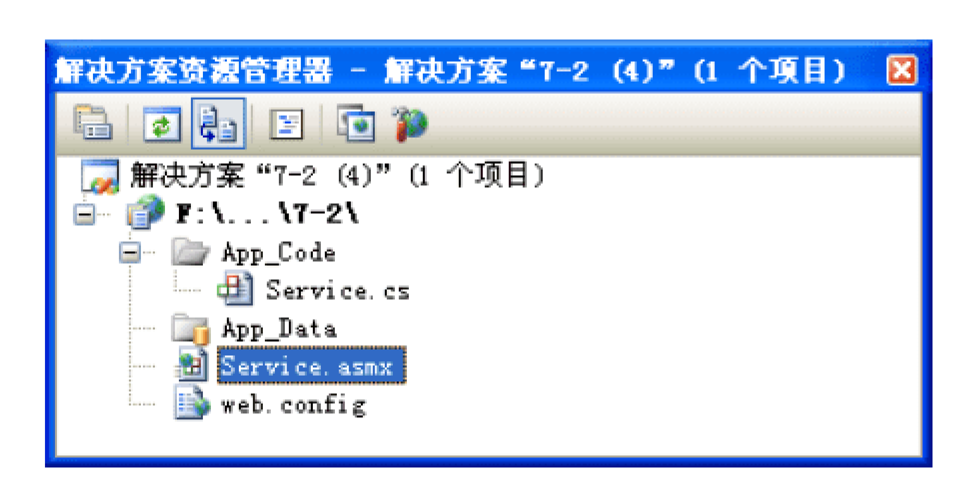


图 8-2　解决方案资源管理器中自动创建的文件和文件夹

解决方案资源管理器中自动生成的 Web 服务的名称为 Service。该服务包含了两个文件，分别为 Service.asmx 和 Service.cs，这两个名字可以更改。但是与其在多处将 Service 更改为其他名字，不如删除文件 Service.cs 和 Service.asmx，再单击【网站】|【添加新项】|【Web 服务】命令来添加一个自定义命名的 Web 服务。如图 8-3 所示，重新添加一个名为 Caculator.asmx 的 Web 服务。

在默认情况下，Visual Studio 2008 以“后台编码”模型创建 Web 服务。在该模型中，.asmx 文件只包含一行，如下所示。



图 8-3　添加名为 Caculator.asmx 的 Web 服务

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Caculator.cs" Class=
"Caculator" %>
```

其中 CodeBehind 属性是 Visual Studio.NET 的专用属性，通过该属性将.asmx 文件与相关的源代码文件相匹配。该属性指定与 Caculator.asmx 文件匹配的源代码文件是 Caculator.cs，如代码 8.16 所示。

代码 8.16　服务：Caculator.asmx

```
using System;
using System.Collections;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;
/// <summary>
///Caculator 的摘要说明
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
//若要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务，请取消对下行的注释。
// [System.Web.Script.Services.ScriptService]
public class Caculator : System.Web.Services.WebService {
    public Caculator () {
        //如果使用设计的组件，请取消注释以下行
        //InitializeComponent();
    }
    [WebMethod]
    public string HelloWorld() {
```

```
            return "Hello World";
        }

    }
```

这些代码都是 Visual Studio 2008 自动生成的，这与使用记事本等文本编辑器编写的 Web 服务代码略有不同，但是两者的核心内容一样。在该服务中，现在能够被使用的方法只有 HelloWorld()方法。由于已经被标识了 WebMethod 属性，调用该方法后，将会返回一个内容为 Hello World 的字符串。

再向 Caculator 类中添加一个 multiplyValue()方法，该方法有两个 Double 类型的参数 Num1 和 Num2，如代码 8.17 所示。

代码 8.17　计算两个双精度数的积：multiplyValue(double Num1, double Num2)

```
[WebMethod]
public double multiplyValue(double Num1, double Num2)
{
    return Num1 * Num2;
}
```

运行网站后，列表中列出了两个方法 HelloWorld()和 multiplyValue()，如图 8-4 所示。

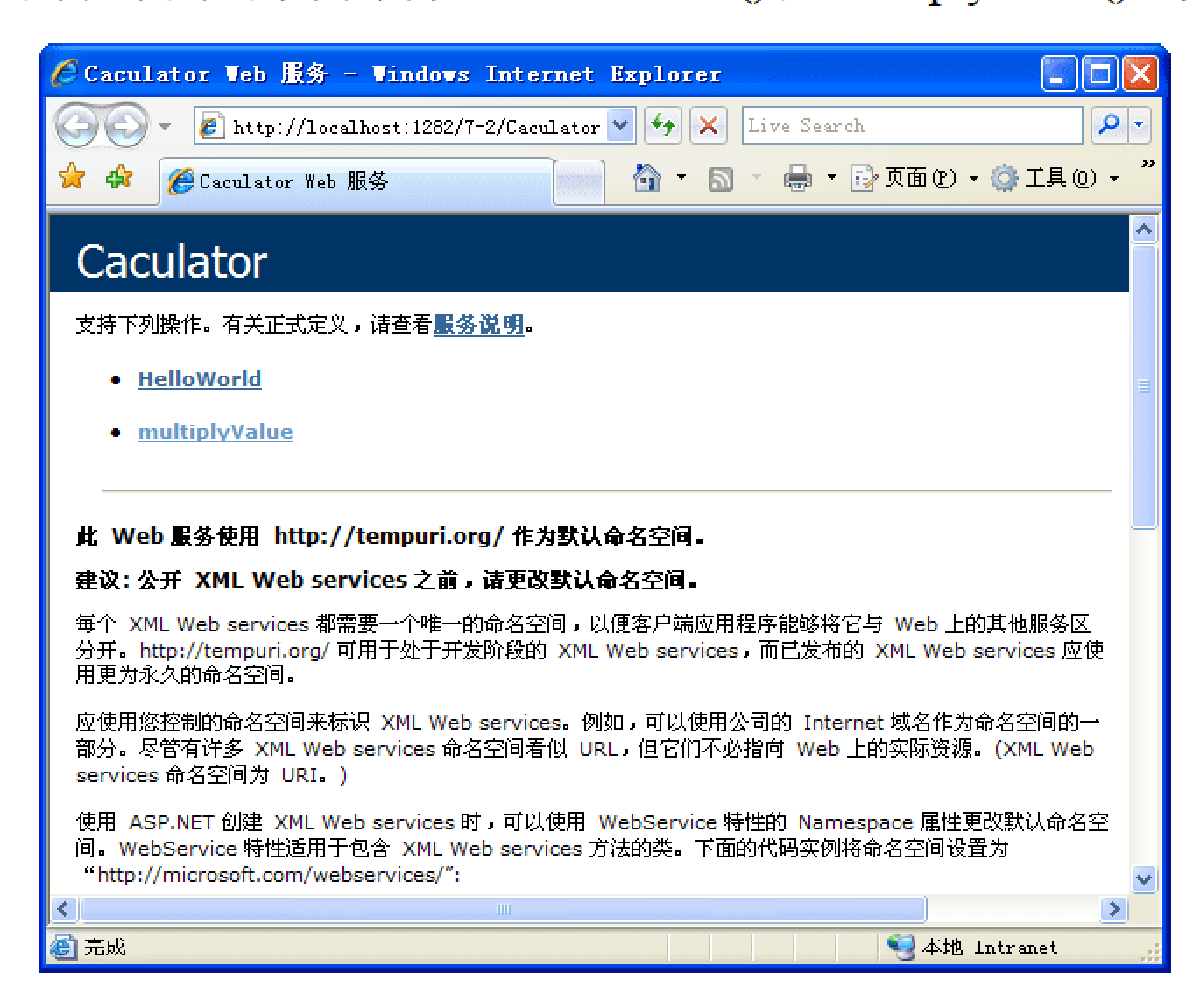


图 8-4　网站运行后 Caculator Web 服务效果

在如图 8-4 所示的页面中，最上面的 Caculator 是 Web 服务的名字，接下来是 Web 服务中可调用的方法，单击 multiplyValue 超链接，弹出效果如图 8-5 所示的页面。

设置 Num1 的值为 123，Num2 的值为 456，单击【调用】按钮，会出现如图 8-6 所示的页面，表示 Web 服务已经创建成功。



图 8-5　测试 multiplyValue()方法

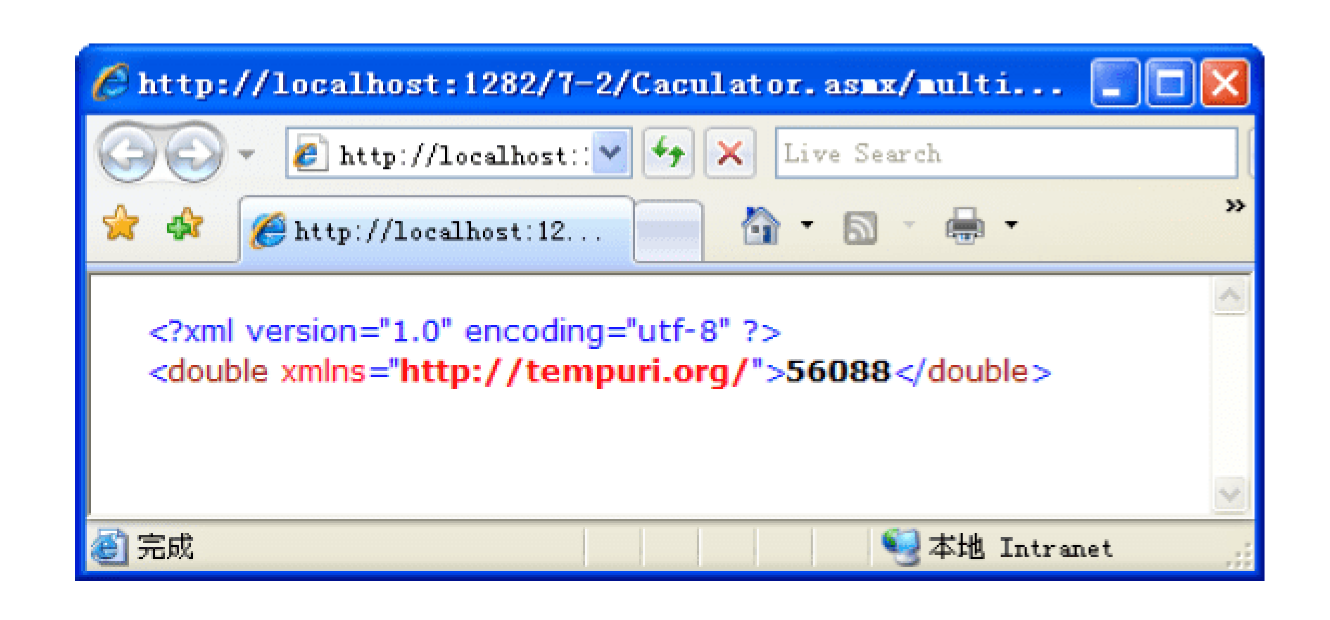


图 8-6　调用 Web 服务后的结果

8.2.3　使用 Web 服务

.NET 客户应用程序可以通过一个代理与 Web 服务进行通信。这个代理只不过是从服务的 WSDL 文档创建的一个.NET 程序集，可以由 Visual Studio 2008 自动创建。

下面在上一节创建的 Web 服务基础上，使用 Visual Studio 2008 添加该服务到网站中，介绍如何在 ASP.NET 网站中使用 Web 服务。

首先，使用 Visual Studio 2008 上一节创建的网站，然后，右击【解决方案资源管理器】的网站名称，选择【添加 Web 引用】弹出【添加 Web 引用】对话框，效果如图 8-7 所示。

单击对话框中【此解决方案中的 Web 服务】超链接，转入显示 Web 服务的页面，效果如图 8-8 所示。

单击 Caculator 超链接，转到显示服务方法页，并设置【Web 引用名】为 CaculatorServices，效果如图 8-9 所示。

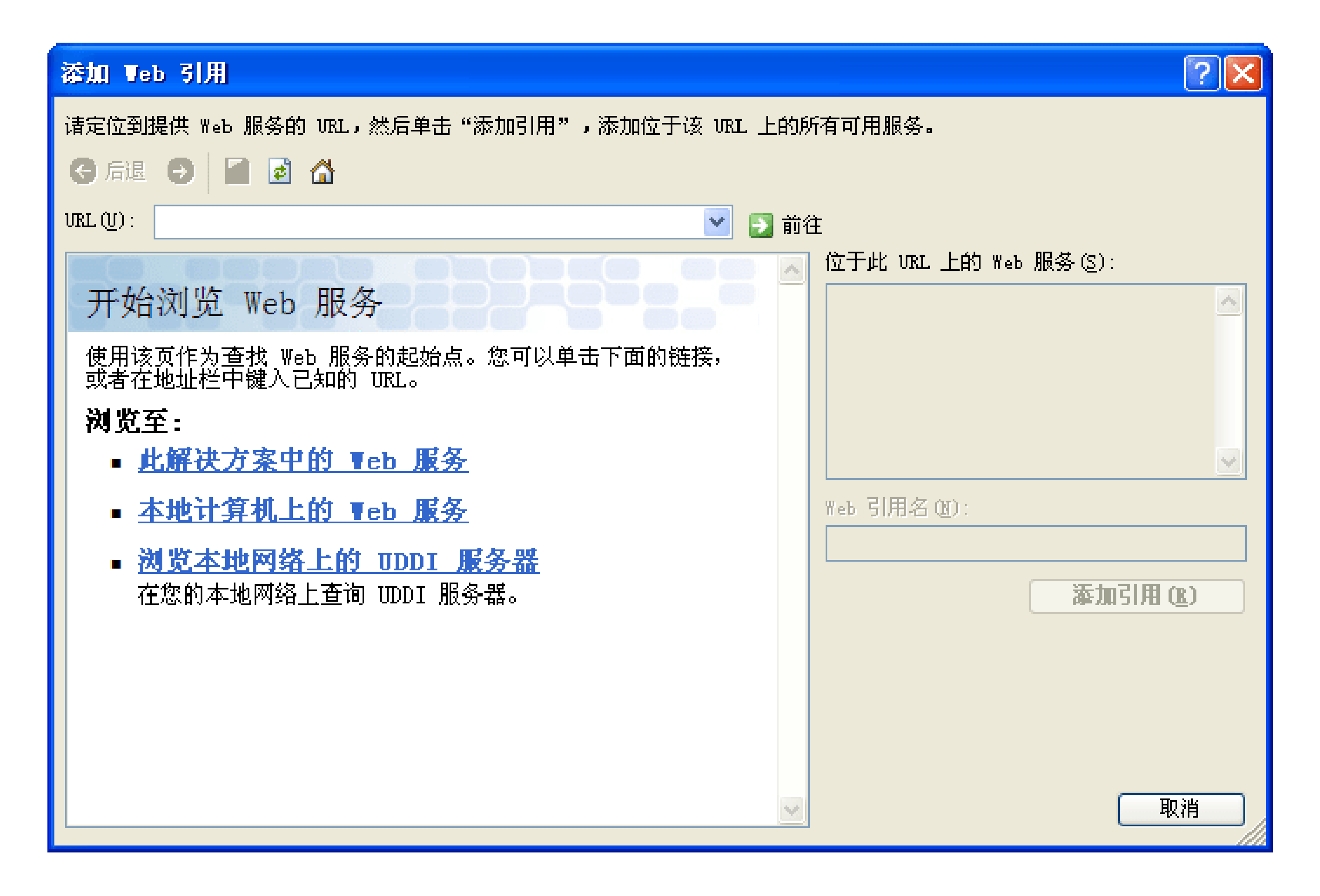


图 8-7 【添加 Web 引用】对话框

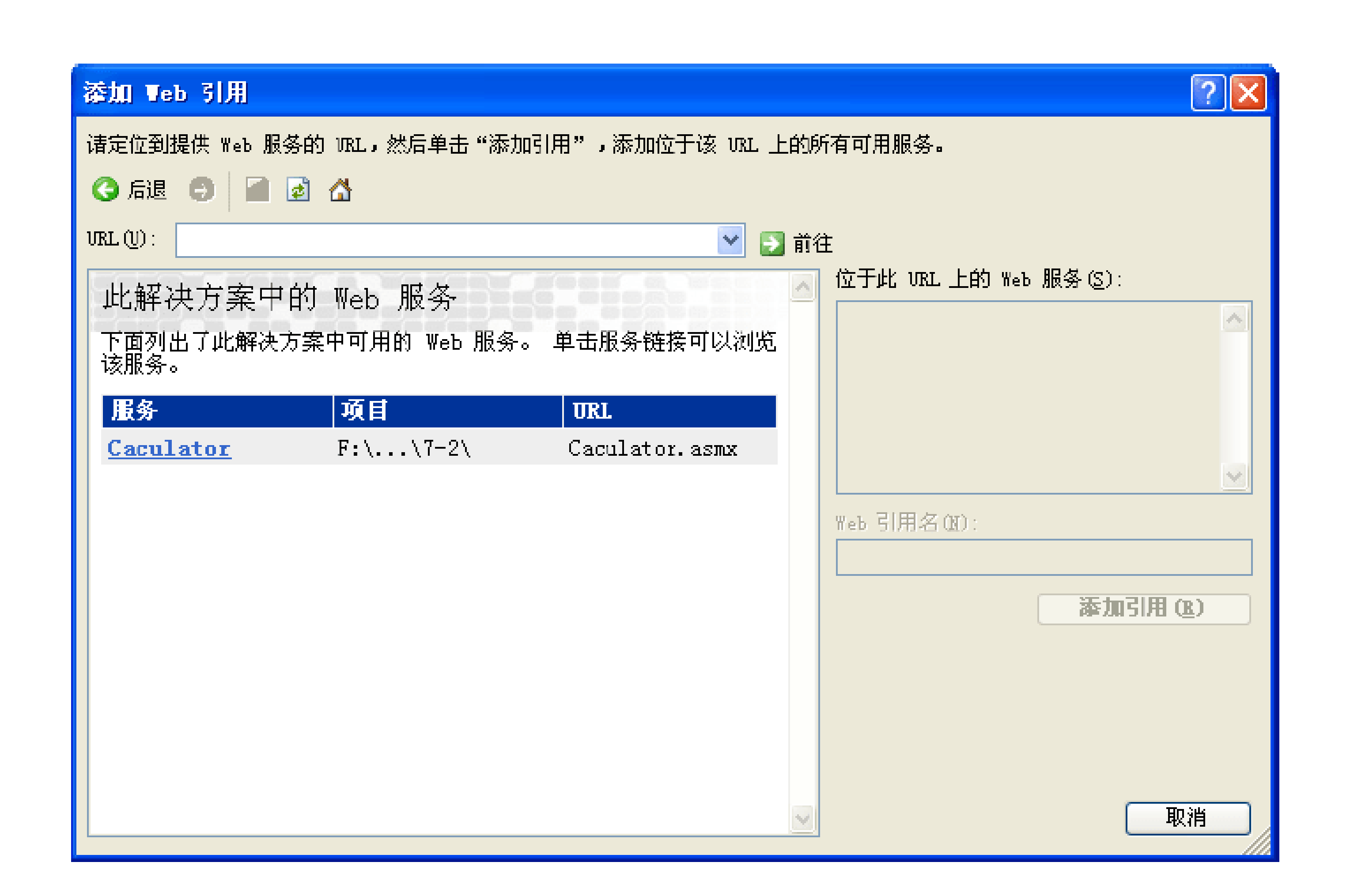


图 8-8 显示 Web 服务的页面

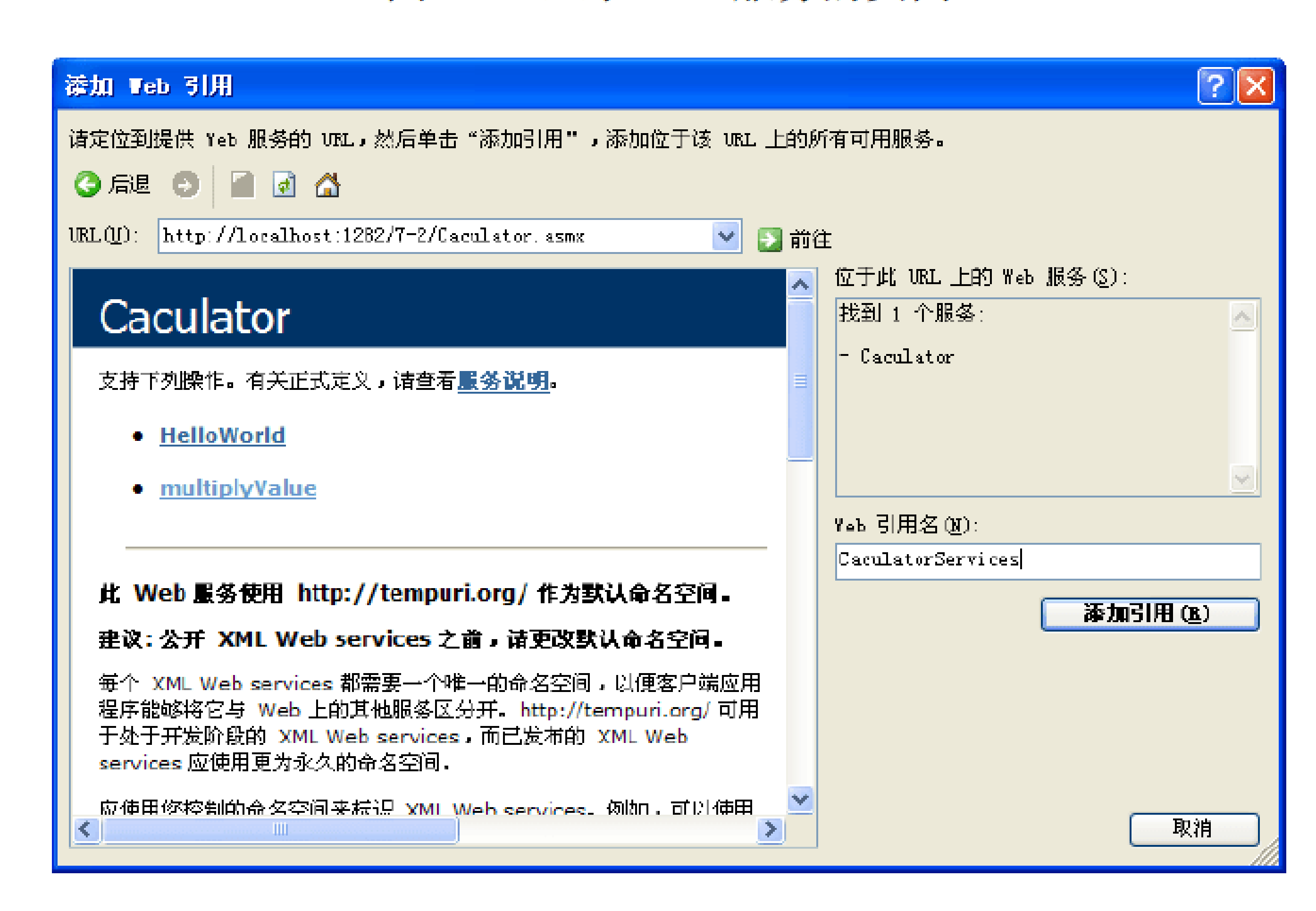


图 8-9 设置 Web 引用名

单击【添加引用】按钮，Web 引用添加成功。此时在解决方案资源管理器中已添加了 App_WebReferences 文件夹以及相应的子文件夹和文件，【解决方案资源管理器】中的内容如图 8-10 所示。

为了在 Default.aspx 页面中显示出调用 Web 服务的结果，向 Default.aspx 页面中添加服务器控件，为用户提供交互界面，具体内容如代码 8.18 所示。



图 8-10　Web 引用添加成功

代码 8.18　添加服务器控件：Default.aspx

```
<asp:Label ID="Label1" runat="server" Text="数字 1: "></asp:Label>
<asp:TextBox ID="tbNum1" runat="server"></asp:TextBox>
<br />
<asp:Label ID="Label2" runat="server" Text="数字 2: "></asp:Label>
<asp:TextBox ID="tbNum2" runat="server"></asp:TextBox>
<br />
<asp:Label ID="lbResult" runat="server"></asp:Label>
<br />
<asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="计
算乘积" />
```

在 Default.aspx.cs 文件中，添加单击【计算乘积】按钮的事件处理函数。该函数使用 Web 服务显示两个数字的乘积，如代码 8.19 所示。

代码 8.19　使用 Web 服务显示两数字的乘积：Default.aspx.cs

```
protected void Button1_Click(object sender, EventArgs e)
{
    double num1 = double.Parse(tbNum1.Text.ToString());
    double num2 = double.Parse(tbNum2.Text.ToString());
    Caculator ca = new Caculator();
    double result = ca.multiplyValue(num1, num2);
    lbResult.Text = "结果: "+result.ToString();
}
```

在代码 8.19 添加完成后运行网站，此时在 Default.aspx 页面中可以输入两个数字，例如 123.123 和 456.456，单击【计算乘积】按钮，效果如图 8-11 所示。

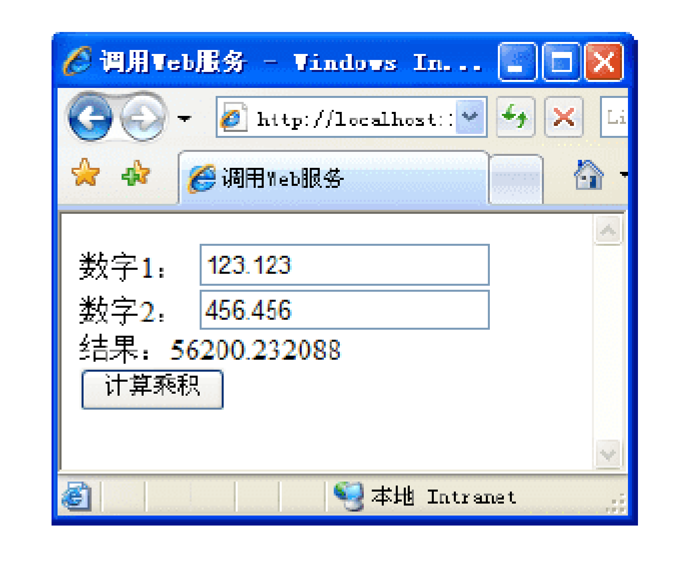


图 8-11　使用 Web 服务计算两数乘

8.3　处理文件

程序开发人员经常需要访问目录和文件，以便收集

信息、对文件系统进行必要的更改。例如，访问文件系统、读取或写入文件、移动或复制文件或浏览文件夹以便检查其中的所有文件。文件可以很方便地用于在应用程序的实例之间存储数据，也可以用于在应用程序之间传输数据。ASP.NET 中可使用 System.IO 命名空间提供的类对驱动器、文件夹和文件进行操作。

8.3.1　System.IO 命名空间

.NET Framework 提供的 System.IO 命名空间，包含允许读写文件和数据流的类型，以及提供基本文件和目录支持的类型。该命名空间提供了基于流的 I/O 操作方式，可以大大简化程序开发者的工作，因为开发人员可以通过对象进行一系列的操作，而不必关心 I/O 操作是和本地计算机中的文件有关，还是和网络中的数据有关。

在.NET Framework 中，System.IO 命名空间主要包含基于文件（和基于内存）的输入输出（I/O）服务的相关基类库。和其他命名空间一样，System.IO 定义了一系列类、接口、枚举、结构和委托。表 8-2 列出了 System.IO 命名空间中与文件管理相关的类及其作用。

表 8-2　与文件管理相关的类及其作用

类名	说明
BinaryReader	用特定的编码将基元数据类型读作二进制值
BinaryWriter	以二进制形式将基元类型写入流，并支持用特定的编码写入字符串
BufferedStream	给另一流上的读写操作添加一个缓冲层。无法继承此类
Directory	公开用于创建、移动和枚举目录和子目录的静态方法
DirectoryInfo	公开用于创建、移动和枚举目录和子目录的实例方法
File	提供用于创建、复制、删除、移动和打开文件的静态方法
FileInfo	提供用于创建、复制、删除、移动和打开文件的实例方法
FileStream	既支持同步读写操作，也支持异步读写操作
MemoryStream	创建以内存作为其支持存储区的流
Stream	提供字节序列的一般视图
StreamReader	以一种特定的编码从字节流中读取字符
StreamWriter	以一种特定的编码向流中写入字符
StringReader	实现从字符串中读取
StringWriter	实现将信息写入字符串
TextReader	表示可读取连续字符系列的阅读器
TextWriter	表示可以编写一个有序字符系列的编写器。该类为抽象类

表 8-2 中包括文件操作的所有类，其中 File 类公开用于创建、复制、删除、移动和代开文件的静态方法，FileInFo 类提供创建、赋值、删除、移动和打开文件的实例方法。通过对两个类的使用，开发人员就可以获得文件的相关信息，同样使用 Directory 和 DirectioryInfo 类可以获得目录的相关信息。

8.3.2　操作驱动器

从.NET 2.0 开始，System.IO 命名空间提供了一个名为 DriveInfo 的类，该类主要用于对计算机上的驱动器进行操作。和 Directory.GetLogicalDrives()相似，DriveInfo.GetDrives()静态方

法能获取计算机上驱动器的名字。然而和 Directory.GetLogicalDrives()不同，DriveInfo 提供了许多其他的细节（比如驱动器类型、可用空间、卷标等）。

DriveInfo 主要包括如下几个与驱动器有关的属性。

- **AvailableFreeSpace**　指示驱动器上的可用空闲空间量。
- **DriveFormat**　获取文件系统的名称，例如 NTFS 或 FAT32。
- **DriveType**　获取驱动器类型。
- **IsReady**　获取一个指示驱动器是否已准备好的值。
- **Name**　获取驱动器的名称。
- **RootDirectory**　获取驱动器的根目录。
- **TotalFreeSpace**　获取驱动器上的空闲空间总量。
- **TotalSize**　获取驱动器上存储空间的总大小。
- **VolumeLabel**　获取或设置驱动器的卷标。

例如，代码 8.20 使用 DriveInfo 类，列出了本机所有驱动器的信息，包括逻辑磁盘的卷标、文件系统类型、容量和可用容量。

代码 8.20　使用 DriveInfo 类

```
protected void Page_Load(object sender, EventArgs e)
{
    DriveInfo[] drives;
    drives = DriveInfo.GetDrives();
    Response.Write("本机器的磁盘信息如下: <br>");
    foreach (var item in drives) {
        Response.Write(item.Name + "<br>");
        switch (item.DriveType) {
            case DriveType.Fixed:
                Response.Write("卷标: " + item.VolumeLabel + "<br>");
                Response.Write("文件系统类型: " + item.DriveFormat + "<br>");
                Response.Write("容量: " + item.TotalSize.ToString() + "<br>");
                Response.Write("可用容量: "+item.AvailableFreeSpace.ToString()
                + "<br>");
                break;
            case DriveType.CDRom:
                Response.Write("该驱动器为光驱<br>");
                break;
            case DriveType.Network:
                Response.Write("该驱动器为网络驱动器<br>");
                break;
            case DriveType.Removable:
                Response.Write("该驱动器为可移动磁盘<br>");
                break;
        }
    }
}
```

图 8-12 所示是代码 8.20 的运行效果，但需要先引入 System.IO 命名空间。



图 8-12　显示驱动器信息

8.3.3　操作文件夹

在 System.IO 命名空间中，提供了 Directory 和 DirectoryInfo 类来进行文件夹（目录）管理。使用它可以完成对文件夹及其子文件夹的创建、移动、浏览等操作，甚至还可以定义隐藏文件夹和只读文件夹。Directory 类的所有方法都是静态的，因此无须创建对象即可调用，这些方法可以操纵和查询任何文件夹的信息。

DirectoryInfo 有实例成员，因此必须利用一个对象引用来访问它们，还能更加有效地对一个文件夹进行多种操作。该类实例化后，就可以保存文件夹的创建时间和最后修改时间等状态。当使用该类的方法时，保存的状态便可以提供结果。

在使用 Directory 类和 DirectoryInfo 类时，必须理解当前工作目录（文件夹）的概念。在应用程序运行时，某一时刻只能有一个当前工作目录。如果对目录或者文件使用相对目录引用，C#将会使用当前目录作为相对引用的开始；如果使用绝对引用，则忽略当前工作目录。还有一点值得注意，在 C#中\是特殊字符，如果表示它的话需要使用\\。由于这种写法不方便，C#语言提供了@对其简化。只要在字符串前加上@即可直接使用\。所以在书写路径时，要么书写\\，要么在路径的前面添加@，例如，@"E:\shugao"。

通过上面的讲述，可以看出这两种类都提供了多种方法，用于对目录进行操作，DirectoryInfo 类还提供了大量的属性。下面列举一些常见的目录操作，并以实例的方式讲解这些方法和 DirectoryInfo 类的一些属性。

1. Exists()方法

Exists()方法接受一个参数，参数表示包含当前工作目录的字符串。返回指示目录是否存在的 bool 值。如果存在，返回 true，否则返回 false。使用 Delete()方法删除目录时，参数所表示的目录一定要存在，否则系统将抛出一个异常。为避免此类异常发生，可以先用 Exists()方法判断目录是否存在。代码 8.21 判断目录 E:\shugao 是否存在。如果存在会弹出提示信息为“该文件夹存在。”的对话框，否则会弹出提示信息为“该文件夹不存在。”的对话框。

代码 8.21　Exists()方法的使用

```
if (Directory.Exists(@"E:\shugao ")) {
    Response.Write( "<script>alert('该文件夹存在。')</script>");
   }
   else  {
      Response.Write("<script>alert('该文件夹不存在。')</script>");
   }
```

2．CreatDirectory()方法

接受一个创建目录路径的 String 类型参数，该方法会根据此路径创建新目录，并且该方法返回一个 DirectoryInfo 实例。如果创建的目录已经存在，则返回代表指定目录的类实例，不会创建目录，也不会产生异常。代码 8.22 演示了如何创建目录。

代码 8.22　CreatDirectory()方法

```
if (!Directory.Exists(@"E:\shugao ")) {
    Directory.CreateDirectory(@"E:\shugao ");
}
```

3．目录属性设置 DirectoryInfo 类的 Attributes 属性

DirectoryInfo 类的 Attributes 属性可以获取或设置当前目录的相关属性。下面创建的代码设置 E:\shugao 目录为只读和系统文件。目录属性和文件属性都是通过 FileAttributes 进行设置的，该实例主要内容如代码 8.23 所示。

代码 8.23　设置目录属性

```
DirectoryInfo dirInfo = new DirectoryInfo(@"E:\shugao ");
dirInfo.Attributes = FileAttributes.ReadOnly | FileAttributes.Temporary;
```

4．Delete()方法

该方法用于删除指定目录，该方法为一个重载方法，可以接受一个参数或者两个参数。当方法接受一个参数时，该参数为指定目录的路径，只能删除指定的空目录，如果目录不为空，系统会抛出异常。当该方法接受两个参数时，第二个参数为 bool 值，如果为 true，则可以删除非空目录。代码 8.24 演示了该方法的使用。

代码 8.24　Delete()方法删除目录

```
if (Directory.Exists(@"E:\shugao "))  {
    Directory.Delete(@"E:\shugao ", true);
}
```

该段代码将会删除目录以及目录中的所有子目录和文件。在删除目录时要谨慎，因为在删除目录时系统不会向用户提供任何确认信息，而且一旦删除就无法恢复。这样用户很有可能把一些有用的目录也给删掉。此时希望了解所要删除目录的一些信息，比如它是否存在子目录或者文件，既该目录是否为空。

5．Move()方法

Move()方法接受两个参数，用于把已有的目录移动到另一个目录。该方法具有两个参数，第一个参数为源目录，第二个参数为目标目录。Move()方法也可以用来重命名目录，可以使用相对目录引用和绝对目录引用，但是不能把目录在不同的逻辑驱动器之间移动。代码 8.25 演示了如何使用 Move()方法移动目录。

代码 8.25　Move()方法移动目录

```
Directory.Move(@"E:\shugao ", @" D:\book ");
Directory.Move(@"E:\shugao ", @" E:\book ");
```

代码展示了使用 Move()方法移动目录和重命名目录。第一行代码把目录 E:\shugao 及其子目录和文件移动到目录 D:\book，不修改目录名。第二行代码把目录 E:\shugao 重命名为 E:\book。

6. GetFiles()方法

重载的 GetFiles()方法具有两种形式，一个参数或两个参数。如果方法采用一个参数时，该参数表示目录名。如果方法采用两个参数时，第一个参数表示目录名，第二个参数表示与第一个参数所包含的文件匹配的文件名，如果存在就返回所匹配文件的绝对目录，否则不返回任何信息。

使用 GetFiles()方法可以返回目录所包含的文件名，代码 8.26 返回绝对目录项目文件夹 images 所包含的文件名。

代码 8.26　GetFiles()方法显示目录所包含的文件

```
protected void Page_Load(object sender, EventArgs e)  {
    string pathString = Server.MapPath("images");
    if (Directory.Exists(pathString))  {
        string[] fileName = Directory.GetFiles(pathString);
        Response.Write("<h3>images 文件列表</h3>");
        for (int i = 0; i < fileName.Count(); i++)  {
            Response.Write(fileName[i] + "<br/>");
        }
    }
}
```

编写完代码 8.26，运行程序，页面效果如图 8-13 所示。

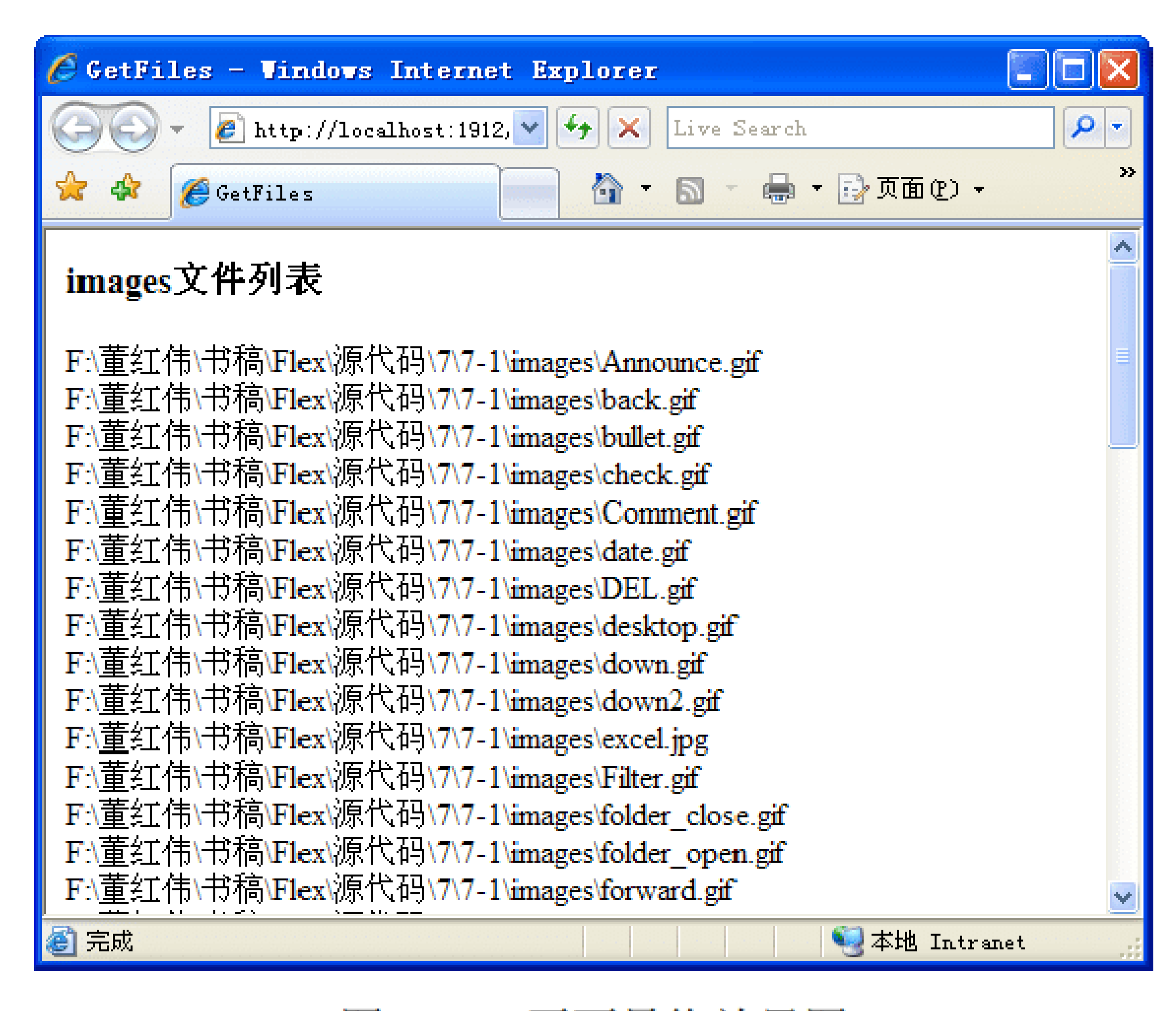


图 8-13　页面最终效果图

7. GetDirectories()方法

重载的 GetDirectories()方法接受一个或两个参数，在两个重载的方法中，第一个参数表示指定目录路径。如果带一个参数，方法返回参数所表示目录的子目录。在接受两个参数的方法中，第二个参数表示与第一个参数所表示目录的子目录匹配的目录名，如果存在就返回所匹配目录的绝对目录，否则不返回任何信息。使用 GetDirectories()方法返回目录的子目录名。代码 8.27 返回绝对目录 C:\Program Files。

代码 8.27　GetDirectories()方法返回子目录

```
protected void Page_Load(object sender, EventArgs e) {
    string pathString = @"C:\Program Files";
    if (Directory.Exists(pathString))  {
        string[] folderName = Directory.GetDirectories(pathString);
        Response.Write("<span style='color:red'>目录名称: </span><br/>");
        for (int i = 0; i < folderName.Count(); i++)  {
            Response.Write(folderName[i] + "<br/>");
        }
    }
}
```

8. GetCreationTime()、GetLastAccessTime()、GetLastWriteTime()和GetLogicalDrives()方法

- **GetCreationTime()方法**　该方法能够接受一个参数，该参数为包含目录的字符串。该方法返回一个日期时间型数值，该时间表示该目录创建的时间和日期。
- **GetLastAccessTime()方法**　该方法接受一个参数，该参数表示文件的路径。该方法返回一个日期时间型数值，该数值表示最后一次访问参数指定文件的时间和日期。
- **GetLastWriteTime()方法**　该方法接受一个参数，即指定路径的目录。该方法同样也返回一个时间，该时间表示用户最后一次修改该目录的时间。
- **GetLogicalDrives()方法**　该方法不接受任何参数，返回一个包含系统上安装的逻辑驱动器的字符串数组。这些驱动器包括硬盘、CD-ROM 和其他磁盘（如移动硬盘）。代码 8.28 演示了这 4 个方法的使用，在页面 Page_Load 事件中编写如下代码。

代码 8.28　获取目录信息及驱动器信息

```
protected void Page_Load(object sender, EventArgs e)  {
    string path = @"C:\Program Files";
    Response.Write("<fieldset><legend>示例演示</legend>");
    Response.Write("该目录创建时间: ");
    Response.Write(Directory.GetCreationTime(path).ToString() + "<br/>");
    Response.Write("该目录最后一次被查看时间: ");
    Response.Write(Directory.GetLastAccessTime(path).ToString() + "<br/>");
```

```
        Response.Write("该目录最后一次修改时间：");
        Response.Write(Directory.GetLastWriteTime(path).ToString() + "<br/>");
        Response.Write("该计算机可用驱动器：");
        string[] drives = Directory.GetLogicalDrives();
        for (int i = 0; i < drives.Count(); i++) {
            if (i != drives.Count() - 1)  {
                Response.Write(drives[i] + "   |  
                ");
            }
            else  {
                Response.Write(drives[i]);
            }
        }
        Response.Write("</fieldset>");
    }
```

此时，该实例成功完成，运行程序，页面效果如图 8-14 所示。

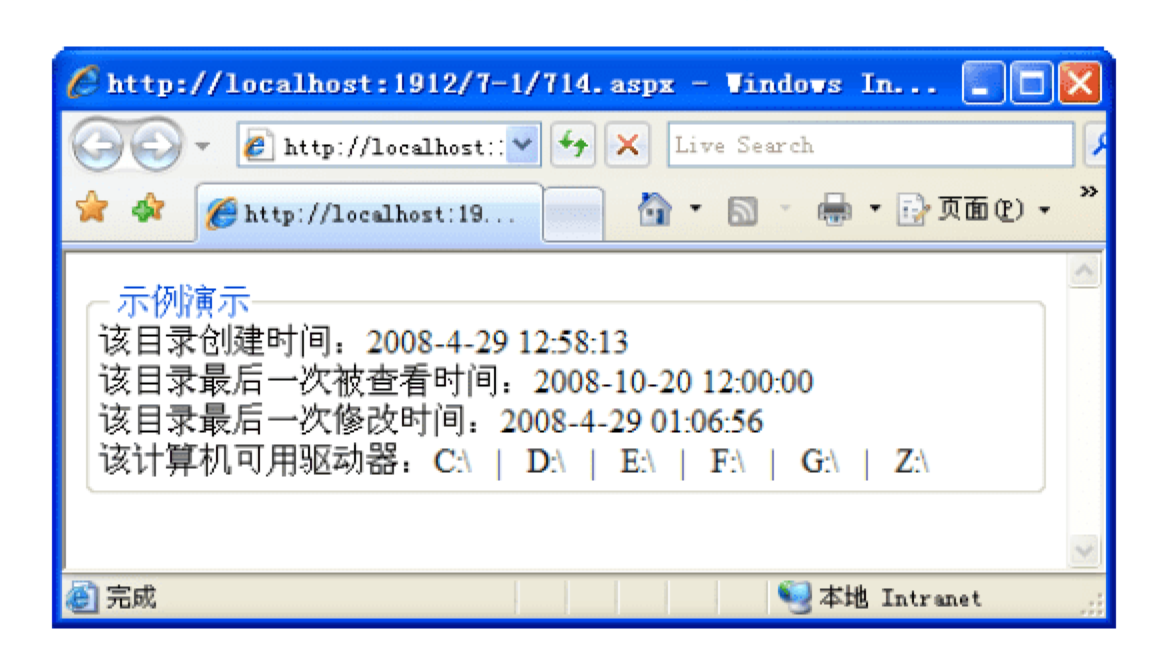


图 8-14　页面效果图

8.3.4　操作文件

File 类的操作方式与 Directory 类相似，它具有用来创建、删除、移动和打开文件等静态方法，如下所示。

- **Create()方法**　该方法为重载方法，接受一个或两个参数。该方法创建一个新的文件，返回 FileStream 类实例。重载方法中，第一个参数表示要创建的文件的目录及文件名；第二个参数是一个整型参数，表示用于读取和写入文件的已放入缓冲区的字节数。如果文件已经存在，将会被重写，C#不会产生异常。因此，要想防止文件被意外重写，可以首先调用 Exists()方法来确保文件不存在。
- **Copy()方法**　该方法为重载方法并接受两个或 3 个参数。该方法把源文件复制到目标文件，可以使用绝对目录和相对目录引用。重载方法中，第一个参数表示源文件，第二个参数表示目标文件。第二种方法有 3 个参数，第 3 个参数为一个 bool 型参数，为 true 时可以改写目标文件，为 false 时不能改写目标文件。

- **Delete()方法**　该方法用于删除文件，可以使用绝对目录和相对目录引用。该方法接受一个参数，表示要删除的文件的目录及文件名。
- **Exists()方法**　该方法具有一个参数，该参数表示文件名。方法返回 bool 值表示文件是否存在，为 true 时表示文件存在，为 false 时表示不存在。
- **Move()方法**　该方法把源文件移动到目标文件。可以使用该方法重命名文件，方法是使用不同的文件名，而源文件和目标文件的目录相同。该方法允许把文件从一个逻辑驱动器移动到另一个逻辑驱动器。该方法接受两个参数，第一个参数表示源文件，第二个参数表示目标文件。
- 每一个文件都具有属性。属性定义了文件的特征，如是否为隐藏文件、只读文件等。GetAttributes()和 SetAttributes()方法分别用于获取和设置文件属性。GetAttributes()方法接受一个参数，表示文件目录及文件名。SetAttributes 方法接受两个参数，第一个表示文件目录及其文件名，第二个参数表示属性。
- **GetCreationTime()、GetLastAccessTime()和 GetLastWriteTime()方法**　这些方法分别用于获取创建文件的日期和时间、最近一次访问文件的时间以及最近一次写入文件的时间。这 3 个方法都接受一个参数，参数表示文件的目录及文件名。

介绍了 File 类的方法，下面就介绍一下如何使用 File 类的方法。代码 8.29 演示了创建、复制、移动和删除文件等操作。

代码 8.29　使用 File 类的方法

```
File.Create("E:\\C#\\file.txt");                              //创建文件
File.Copy("E:\\C#\\file.txt","E:\\C#\\fileinfo.txt");         //复制文件
File.Move("E:\\C#\\file.txt","E:\\C#\\File\\file.txt");       //移动文件
File.Move("E:\\C#\\file.txt","E:\\C#\\File.txt");             //重命名文件
File.Delete("file.txt");                                      //删除文件
```

第一行代码使用绝对目录引用在 E:\C#中创建文件 file.txt。第 2 行、第 3 行和第 4 行代码分别使用绝对目录引用复制、移动和重命名文件。最后一行代码从当前工作目录中删除文件 file.txt。

8.3.5　读写文件

在了解了 System.IO 命名空间中对驱动器、文件夹和文件的操作后，本节将介绍如何对文件进行读取和写入操作。这主要是使用了 System.IO 中的 StreamReader 和 StreamWriter 类。

1．读取文件

读取文件可以使用 StreamRader 类来轻松完成，该类可以处理任何流信息。StreamReader 旨在以一种特定的编码从字节流中读取字符，它的主要方法如下：

- **Close()方法**　该方法关闭与 StreamReader 实例相关的文件。文件读取之后应该显式关闭。
- **Read()方法**　该方法返回一个整数并提升字符的位置，如果没有可用字符则返回–1。

- **ReadLine()方法** 该方法返回文件中的下一行，或者如果到达了文件的末尾，则为空引用。
- **ReadToEnd()方法** 该方法返回从文件的当前位置到文件结尾的字符串。如果当前位置为文件头，则读取整个文件。
- **Peek()方法** 该方法返回文件的下一个字符，但并不使用它。如果没有可用的字符或者文件不支持查找，则返回–1。

使用 StreamReader 类的步骤为：先声明一个 StreamReader 类的对象。构造函数声明了文件和可选路径。这一步将打开文件，这时可以从文件中读取数据。然后调用 StreamReader 类的方法读取数据。最后完成后调用 StreamReader 类的 Close 方法关闭流。代码 8.30 创建一个 StreamReader 类读取文件的实例，来具体讲解该类如何读取文件。该实例比较简单，具体代码如下所示。

代码 8.30　从文件中读取数据

```
protected void Page_Load(object sender, EventArgs e) {
    if (!IsPostBack)
    {
        string pathString = Server.MapPath("Message.txt");
        if (File.Exists(pathString))
        {
            FileStream fileStream = File.OpenRead(pathString);
            try
            {
                StreamReader reader = new StreamReader(fileStream, System.
                Text.Encoding.Default);
                    Response.Write("<table width='100%'>");
                    int s = 0;
                    do
                    {
                        s++;
                        if (s % 2 == 0)
                        {
                        Response.Write("<tr><td align='left' style='back ground-
                        color:#dedede' height='25px'>" + reader.Read Line() +
                        "</tr></td>");
                    }
                    else
                    {
                        Response.Write("<tr><td align='left' height='25px'>"+
                        reader.ReadLine() + "</tr></td>");
                    }
                }
                while (!reader.EndOfStream);
                Response.Write("</table>");
```

```
                    reader.Close();
                }
                catch (Exception ex)
                {
                    this.ClientScript.RegisterStartupScript(this.GetType(), "",
                    "<script>alert(" + ex.Message + ")</script>");
                }
            }
        }
    }
```

在代码 8.30 中，有一点要说明，如果不为 StreamReader()添加第二个参数，当读取的时候会出现乱码，这是由字符集的差异造成的。.NET 默认用 UTF-8 字符集，而系统中的文本文件一般用的是 ANSI 字符集。

使用实例中的方法实例化 StreamReader 类，就可以避免乱码出现。这样实例化 SteamReader 实际上就是选择 StreamReader 默认的字符集，而不是用 UTF-8。这样打开的文件就不会出现乱码。当然，如果文本文件使用 UTF-8 字符集，那么开发人员就不必指定字符集。至此，该实例开发完成，运行程序，页面效果如图 8-15 所示。

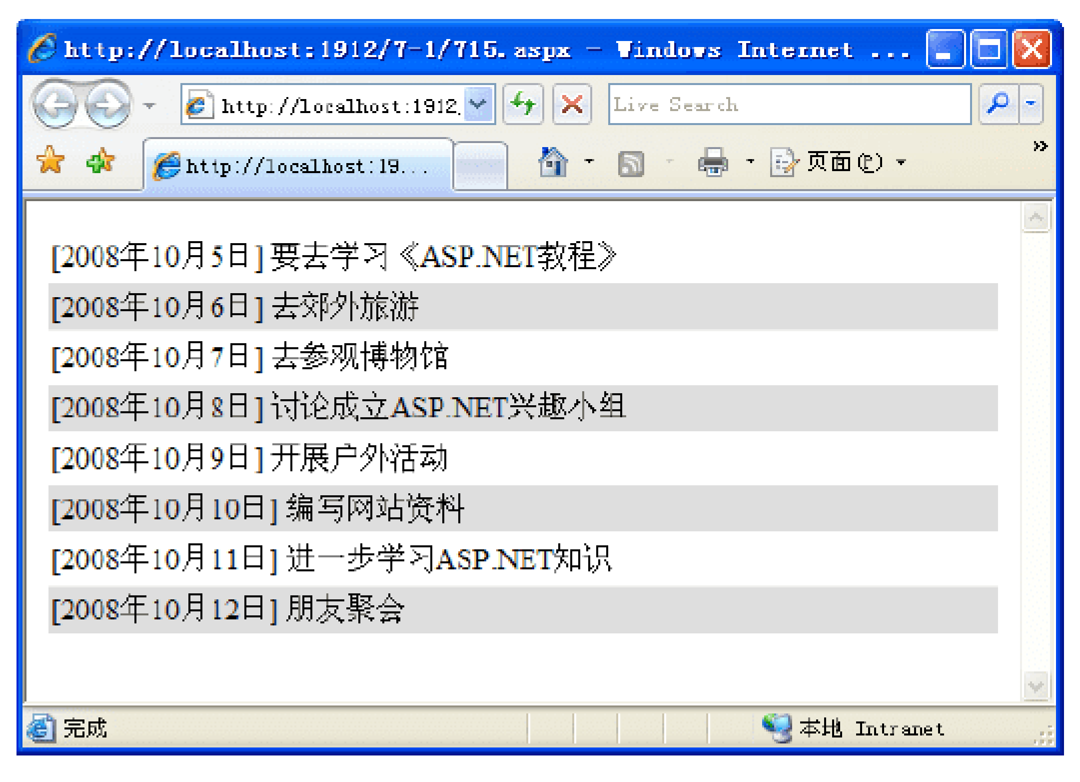


图 8-15　显示从文件中读取的数据

2. 写入文件

一般情况下，向文件中写入内容都是由 StreamWriter 类进行操作，StreamWriter 类以一种特定的编码向字节流中写入字符。与 StreamReader 类相同，StreamWriter 类也具有多个常用的公共方法和属性，它们分别如下所示。

- **Close()方法**　该方法关闭与 StreamWriter 实例相关的文件。在写入文件之后应该显式关闭文件以防止数据丢失。
- **Write()方法**　该方法将字符串写入文件。
- **WriteLine()方法**　该方法向文件写入一行字符，也就是说在文件中写入字符串并换行。
- **Flush()方法**　该方法清理当前编写器的所有缓冲区，并将缓冲区数据写入文件。
- **AutoFlush 属性**　获取或设置一个值，该值指示 StreamWriter 是否在每次调用 Stream

Writer.Write()之后，将其缓冲区刷新到基础流。

- **BaseStream 属性** 获取同后备存储区连接的基础流。
- **Encoding 属性** 获取将输出写入到其中的 Encoding。
- **FormatProvider 属性** 获取控制格式设置的对象。
- **NewLine 属性** 获取或设置由当前 TextWriter 使用的行结束符字符串。

下面创建一个写入文本的实例，展示 StreamWriter 类实例化后方法的使用。创建一个 ASP.NET Web 应用程序，在 Page_Load()方法中添加代码，这里以前面读取文件为基础，为该文件添加新内容，如代码 8.31 所示。

代码 8.31 写入文本

```
protected void Page_Load(object sender, EventArgs e) {
    if (!IsPostBack)  {
        string pathString = Server.MapPath("Message.txt");
        if (System.IO.File.Exists(pathString)) {
            StreamWriter writer = new StreamWriter(pathString, true, System.
            Text.Encoding.Default);
            writer.WriteLine("[2008 年 10 月 20 日] 学习 Flex 第一章");
            writer.WriteLine("[2008 年 10 月 21 日] 学习 Flex 第二章");
            writer.WriteLine("[2008 年 10 月 22 日] 学习 Flex 第三章");
            writer.Flush();
            writer.Close();
            try  {
                FileStream fStream = File.OpenRead(pathString);
                StreamReader reader = new StreamReader(fStream, System. Text.
                Encoding.Default);
                Response.Write("<table width='100%'>");
                int s = 0;
                do  {
                    s++;
                    if (s % 2 == 0) {
                        Response.Write("<tr><td align='left' style='back ground-
                        color:#dedede'>" + reader.ReadLine() + "</tr></td>");
                    }
                    else {
                    Response.Write("<tr><td align='left' >" + reader.ReadLine
                    () + "</tr></td>");
                    }
                }
                while (!reader.EndOfStream);
                Response.Write("</table>");
                reader.Close();
            }
            catch (Exception ex) {
                this.ClientScript.RegisterStartupScript(this.GetType(), "",
```

```
                        "<script>alert(" + ex.Message + ")</script>");
                    }
                }
            }
        }
```


代码 8.31 展示了在文件中追加数据的方法。该实例首先初始化一个 StreamWriter 类对象 swriter，然后调用 StreamWriter 类中的方法 WriteLine()向文件中追加数据，最后调用 StreamWriter 类的 Close()方法关闭文件。然后运行代码，页面效果如图 8-16 所示。

该实例实现了添加文件内容，被添加的内容被追加在原来文件内容的后面。如果将代码 8.31 中创建的 StreamWriter 类中的方法 true 改为 false，新添加的内容就会覆盖原来文件中的内容，更改后的效果如图 8-17 所示。

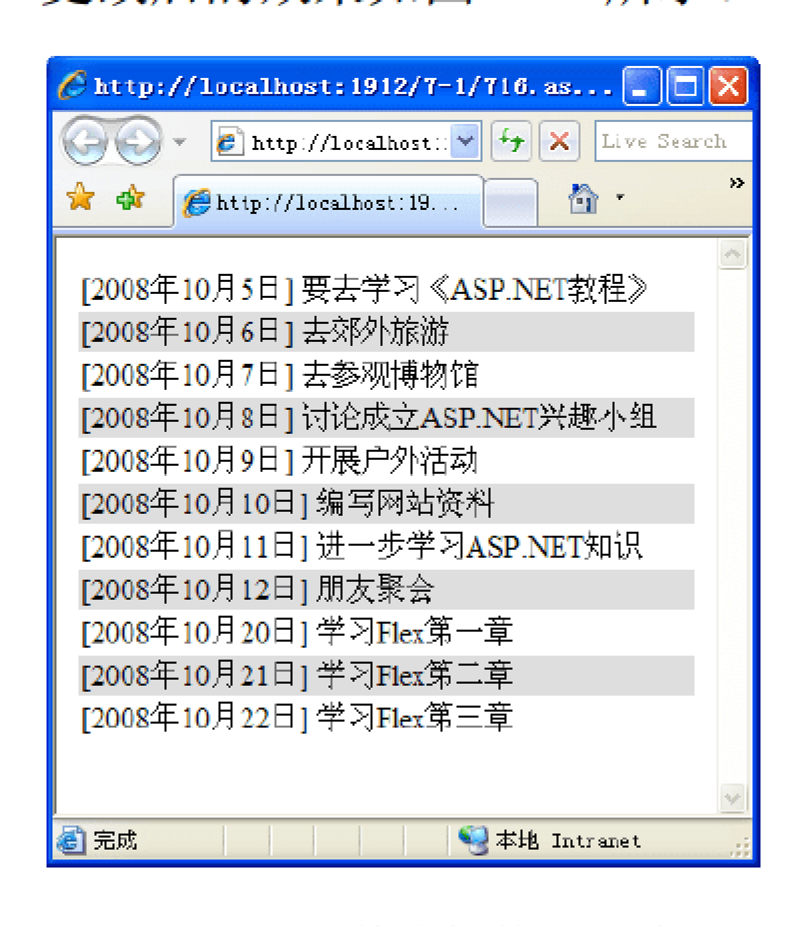


图 8-16　文件追加数据后效果

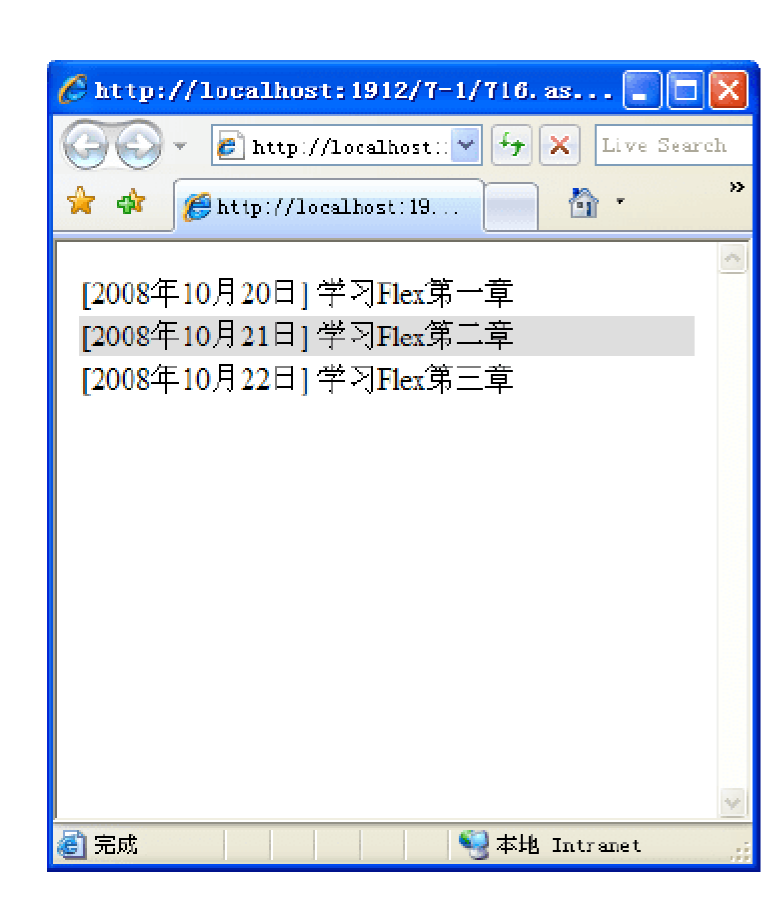
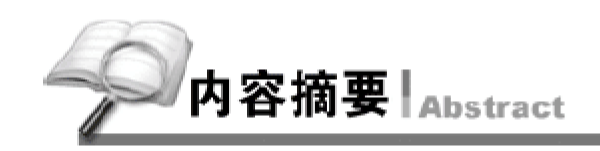


图 8-17　页面效果图

第 3 篇　Flex 组件应用篇

第9章 使用组件



在 Flex Builder 3 中，组件主要包括其内置的可视化组件，如按钮、文本框等。使用这些组件可以加快应用开发的速度。另外，还可以利用 Flex 提供的扩展功能，定制个性化的界面。本章将详细介绍 Flex Builder 3 中提供的各种 Flex 常用组件和导航类组件，并通过实例来演示这些组件的具体应用，为基于组件的应用开发打下基础。



- 了解 Flex 组件
- 掌握各种常用组件的属性及事件
- 熟悉使用 Flex 常用组件设计界面
- 掌握各种导航类组件的使用
- 掌握导航类组件设计界面的方法

9.1 Flex 组件概述

Flex 应用程序的界面通常由各种各样的组件来构建。例如一个登录框应用程序包括了标签组件、输入框组件、按钮组件等。不同的组件具有特定的功能，如按钮组件的单击动作和双击动作。用户可以选择不同的组件来满足应用程序的要求。

组件是能够完成某种功能并且向外提供若干个使用这种功能的接口的可重用代码集，表现形式为常见的库和包。组件将一些类和接口组织起来，对外暴露一个或多个接口，供外界调用。

组件可以将应用程序的前台设计过程和后台编码过程分开。通过使用组件，开发人员可以将常用功能封装到组件中，而设计人员可以通过更改组件的参数来自定义组件的大小、位置和行为。通过编辑组件的图形元素或外观，还可以更改组件的外观。

在 Flex Builder 3 中，可以将所有的组件按照功能划分为以下 4 类。

- **常用控制组件**　在 Flex Builder 3 中，常用控制组件包含了程序开发过程中大部分表单元件，如 Button 组件、Image 组件等。
- **导航类组件**　在 Flex Builder 3 中，导航类组件完成对程序开发的导航功能，如 MenuBar 组件、TabBar 组件等。

- 布局组件　在设计程序界面时，为了界面的美观，需要对界面进行合理的布局。在 Flex Builder 3 中提供了这样一组布局组件，可以帮助用户完成界面的布局，如 Panel 组件、Form 组件等。
- 图表类组件　在 Flex Builder 3 中使用自带的图表类组件可以创建许多不同类型的图表，以更直观的方式展现数据。

Flex Builder 3 中的所有组件都位于 Components 窗格中，并按照其功能划分成为 4 个不同的类型。展开某类型节点，就可以看到该类型下的所有可用组件，如图 9-1 所示。

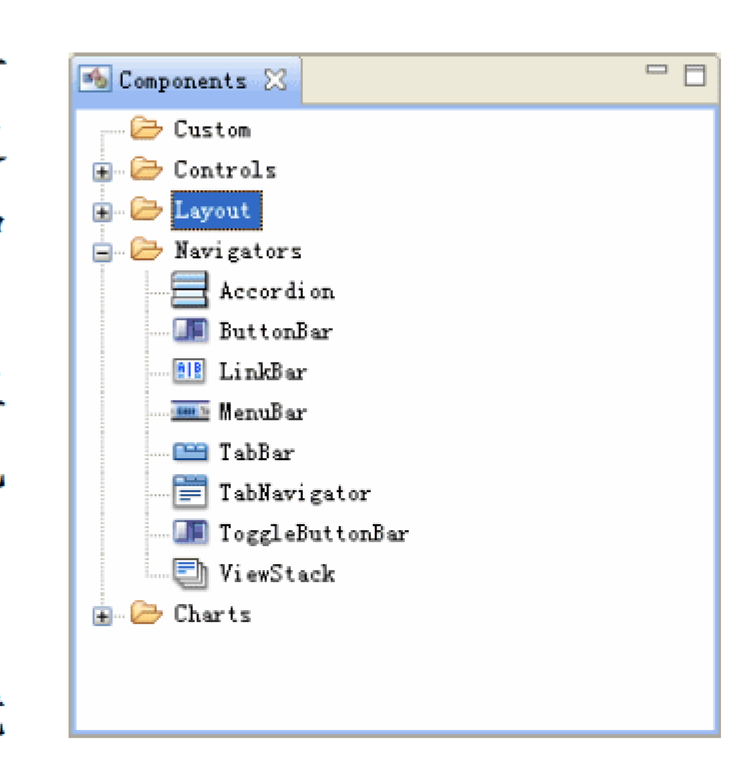


图 9-1　Components 窗格

9.2　Flex 常用组件

在前面的内容中已经和部分组件打过交道，如 Label、Button 组件等。从功能和外观上来看，这些组件和 Flash CS3 或者更高版本中的同名组件并没有太大差异。即使读者没有 Flash 的开发经验，也完全不用担心，在 Flex 中组件的操作非常简单。本节将首先介绍 Flex 中一些常用组件的具体应用。

9.2.1　文本组件

文字处理是程序中必不可缺的部分。在 Flex Builder 3 中提供了一系列的文本组件，包括 Text 组件、TextInput 组件、TextArea 组件和 RichTextEditor 组件。其中在前面反复出现的 Label 组件也属于文本组件，它与 Text 组件功能非常相似。下面将分别介绍这几种组件的属性和应用。

1. Label 组件

Label 组件通常用于提供其他组件的描述性文字，或是显示静态文本。例如，可以使用 Label 为 TextInput 组件添加描述性文字，以便将组件中所需要输入的数据内容通知给用户。

Label 组件在页面上不可以接受焦点，Label 组件有几个比较常用的属性，id、text 和 visible 属性。id 属性唯一标识该 Label 组件，text 属性表示在该组件上的显示文字，visible 属性表示该组件的可见性。

除了上述的属性外，Label 组件还可以通过设置其他属性，如改变显示文本的字体、颜色、对齐方式等。例如下面的代码，就设置字体大小为 16，字体颜色为#C1391A，并且居中对齐。

```
<mx:Label x="57" y="92" text="请输入你的年龄" width="123" id="L1" fontSize="16"
color="#C1391A" textAlign="center"/>
```

2. Text 组件

在 Flex Builder 3 中，Text 组件的功能与 Label 组件非常接近，其属性和 Label 组件的属性

也完全相似。这里就不再重复。

3. TextInput 组件

TextInput 组件用于接受用户的输入文本，与 ASP.NET 中的 TextBox 组件功能相似。表 9-1 中列出了 TextInput 组件的常用属性。

表 9-1　TextInput 组件的常用属性

属性	说明
id	标识唯一的 TextInput 组件
text	TextInput 组件的显示文本
maxChars	TextInput 组件可输入的最大字符数
displayAsPassword	以密码形式显示用户输入的文本
editable	表示是否可对该组件的文本进行编辑
enabled	表示该组件是否可用
visible	表示该组件是否在页面中显示
htmlText	以 HTML 编码形式显示文本内容

和 Label 组件一样，TextInput 组件也可以通过其他属性设置其显示样式，如文本大小、字体颜色等。例如下面一段代码就定义了一个 TextInput 组件。

```
<mx:TextInput x="57" y="59" id="tp1" displayAsPassword="true" maxChars="30"
editable="true" fontFamily="Arial" fontSize="16" color="#1571E2"/>
```

除了使用属性外，Flex 还为 TextInput 组件提供一系列的事件，其中比较常用的事件有：change 事件、click 事件、focusIn 事件和 focusOut 事件。change 事件当 TextInput 组件中的文本发生改变时被调用，click 事件当用户单击 TextInput 组件时被调用，focusIn 事件和 focusOut 事件分别当 TextInput 组件获取和失去焦点时被调用。下面来创建一个简单的实例，具体演示这几种事件的作用，如代码 9.1 所示。

代码 9.1　使用 TextInput 组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"layout="absolute">
<mx:Script>
    <![CDATA[
        import mx.controls.Alert;
        private function show():void
        {
         L3.text="";
         var str:String="你的姓名为: "+TP1.text;
         str +="\n\n 你的密码为: "+TP2.text;
         if(TP1.text!="" && TP2.text!="")
         {
         Alert.show(str);
         }
```

```
            else
            {
            Alert.show("用户名称或者密码为空，请确认");
            TP2.focusEnabled=false;
            }
           }
        ]]>
    </mx:Script>
        <mx:Panel layout="absolute" title="用户注册" fontSize="15" horizontalAlign=
        "center" verticalAlign="middle" id="P1" width="349" height="258" horizon-
        talCenter="12" verticalCenter="12">
            <mx:Label x="24" y="10" width="217" fontSize="15" id="L1"/>
            <mx:TextInput x="24" y="40" fontSize="10" width="168" id="TP1" focusIn=
            "L1.text='请输入你的姓名'" />
            <mx:Label x="24" y="68" width="217" fontSize="15" id="L2"/>
            <mx:TextInput x="24" y="99" width="168" fontSize="10" id="TP2" display
            AsPassword="true" focusIn="L2.text='请输入密码'" change="L3.text='你
            输入的密码为: '+TP2.text" focusOut="show()" />
            <mx:Label x="24" y="130" fontSize="10" width="217" height="36" id=
            "L3"/>
        </mx:Panel>
    </mx:Application>
```

在代码 9.1 中，创建了一个简单的注册界面，并且初始 3 个 Label 组件的 Text 属性为空，当两个文本框获得焦点时，调用 focusIn 事件，显示相应的提示信息。设置第二个文本框的 displayAsPassword 属性为 true，以密码形式显示文本，当文本框中的文字改变时，调用 change 事件，显示用户输入的密码，如图 9-2 所示。最后设置当第二个文本框失去焦点时，调用函数 show()，显示注册信息，如图 9-3 所示。

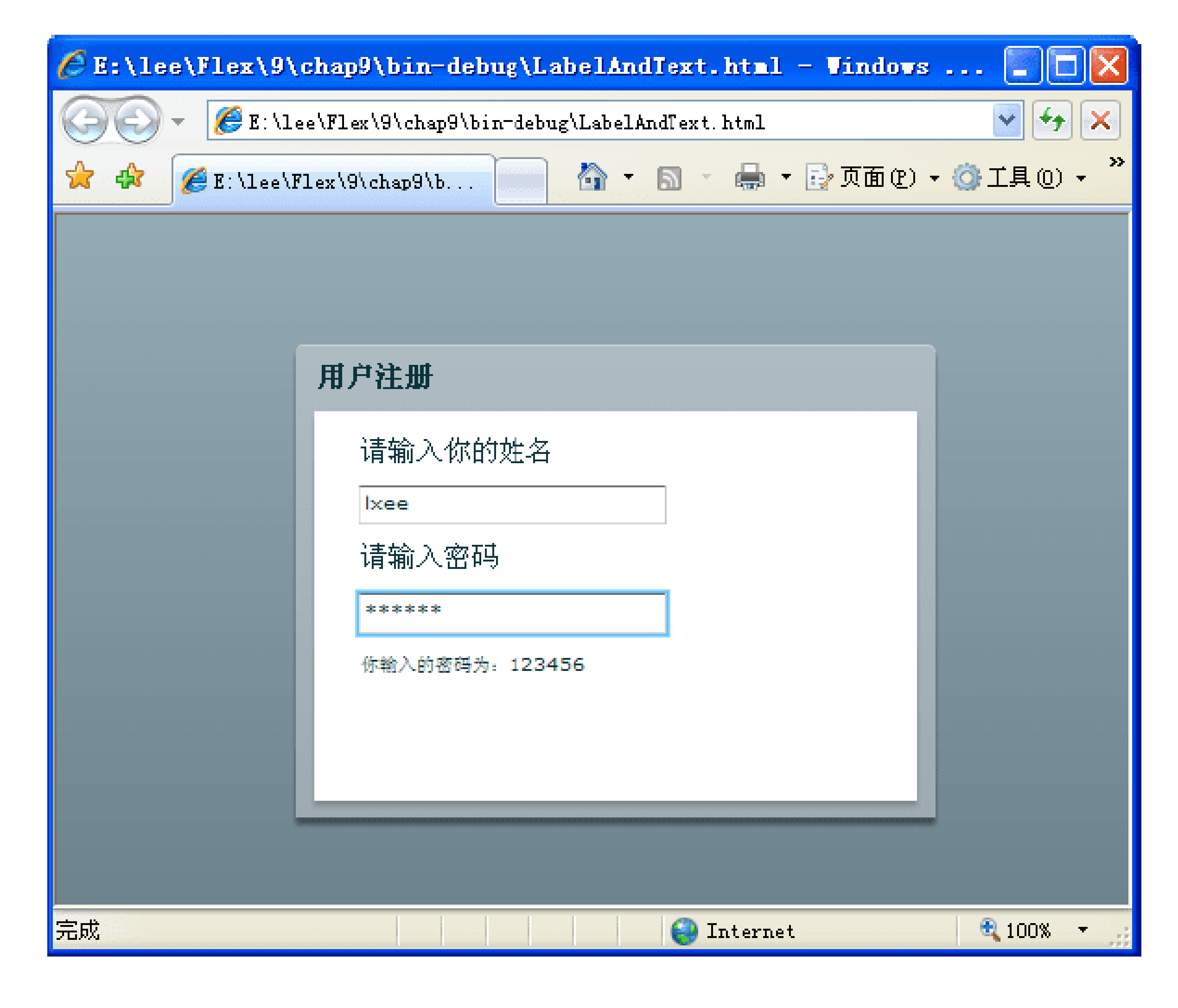


图 9-2　显示提示信息和密码

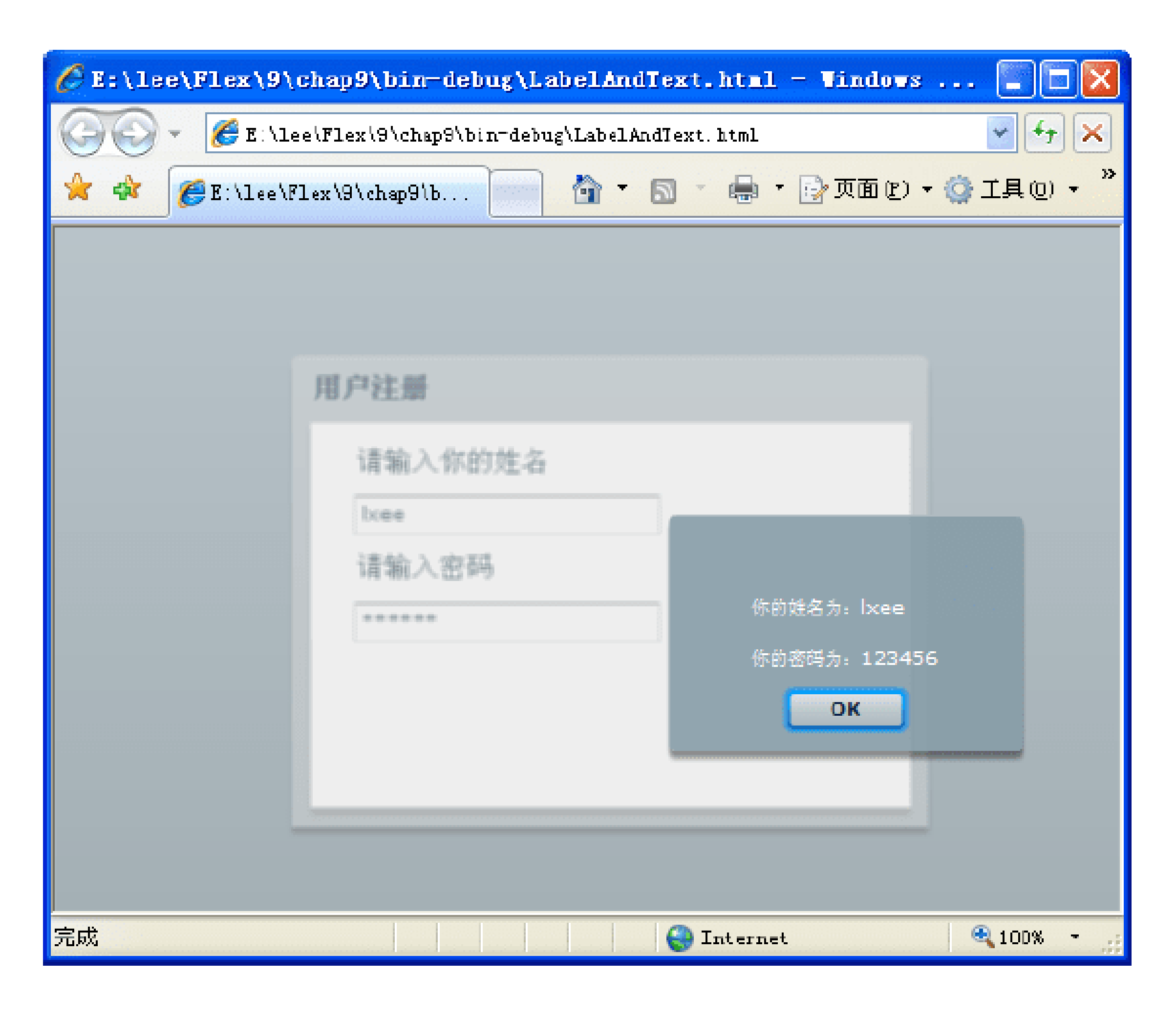


图 9-3　显示注册信息

4. TextArea 组件

在页面中，用户可以使用 TextArea 组件来输入多行文本。TextArea 组件的属性与 TextInput 组件的属性相似。除了表 9-1 列出的那些属性外，TextArea 组件还有一个常用的属性 wordWrap。该属性提示用户在输入的文本超出文本框的宽度范围时是否自动换行，默认选择 true，即超出范围自动换行，如果用户设置该属性为 false，则不会自动换行。

TextArea 组件的事件与 TextInput 组件的事件完全相似，也可以在文字发生改变时或者获得、失去焦点时触发事件，这里不再重复。

5. RichTextEditor 组件

在 Flex Builder 3 中提供了一个功能强大的文本组件 RichTextEditor，该组件实现了 HTMl 标签的所有功能。RichTextEditor 组件是一个集成了文本处理功能的 Panel 组件，主要包含两部分：一个 TextArea 文本区和一个控制文字格式的容器。

在控制文字格式的容器中，又包含了其他的组件，如用于选择字体的下拉菜单、选择文本颜色的 ColorPicker、选择文本对齐方式的 ToggleButtonBar 组件等。

下面就创建了一个简单的实例，在该实例中，展示了 RichTextEditor 组件的一些特性，具体内容如代码 9.2 所示。

代码 9.2　使用 RichTextEditor 组件

```
<?xml version="1.0" ?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   horizontalAlign="center" verticalAlign="middle">
   <mx:RichTextEditor id="rte" title="RichTextEditor" height="75%"
text="Enter text into the RichTextEditor control, then click a button to display
your text as plain text, or as HTML-formatted text."/>
   <mx:TextArea id="rteText" width="80%" height="25%"/>
```

```
        <mx:HBox>
            <mx:Button label="Show Plain Text" click="rteText.text=rte.text;"/>
            <mx:Button label="Show HTML Markup" click="rteText.text=rte.html
            Text;"/>
        </mx:HBox>
    </mx:Application>
```

运行效果如图 9-4 所示。

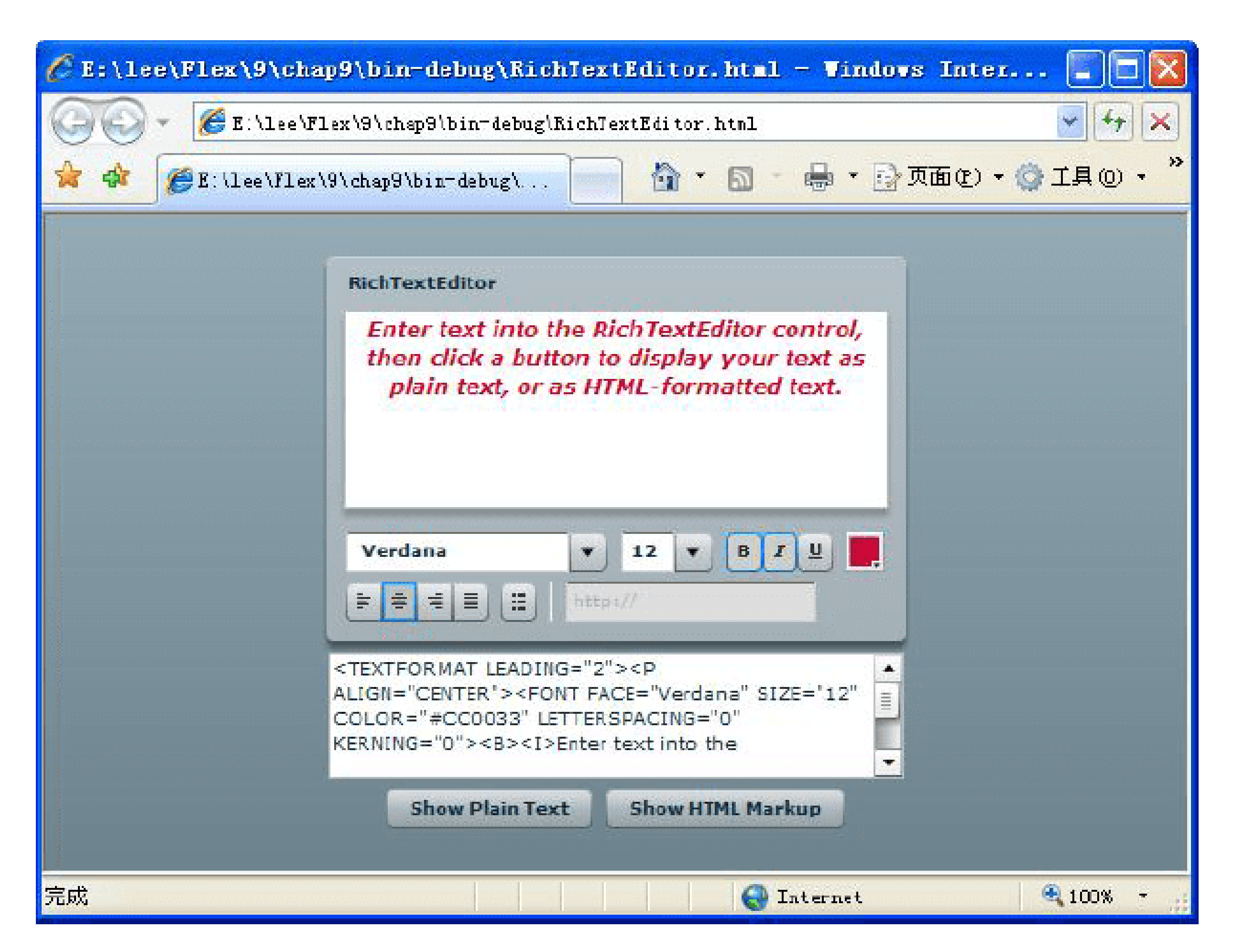


图 9-4　使用 RichTextEditor 组件

选择文本框中的部分文字，改变颜色、大小和对齐方式等。单击下面两个按钮，就可以查看 RichTextEditor 组件中的文本和 HTML 代码。

利用 RichTextEditor 组件，就不需要手动写 HTML 代码，可以利用它实现效果后，直接将代码复制到想用的地方，非常方便。

9.2.2　CheckBox 和 RadioButton 组件

CheckBox 和 RadioButton 组件是在应用程序中经常用到的组件，可以用这两种类型的组件将复选框和单选按钮添加到页面上。

1. CheckBox 组件

CheckBox 组件提供了一组可供用户选择的选项，根据用户的选择获取相对应的信息。该组件的属性如表 9-2 所示。

表 9-2　CheckBox 组件的属性

属性	说明
id	唯一标识一个 CheckBox 组件
label	CheckBox 组件显示的文本
labelPlacement	可以通过设置 labelPlacement 的 top、bottom、left 和 right 参数来实现想要的标签显示效果

续表

属性	说明
selected	CheckBox 组件是否被选择
styleName	CheckBox 组件的样式名称
visible	表示该 CheckBox 组件是否可见
enabled	表示该 CheckBox 组件是否可用
includeInlayout	指定此组件是否包含在父容器的布局中

在 Flex Builder 3 中为 CheckBox 组件定义了一系列的事件，其中最常用的事件是 click 事件，当用户单击选择该复选框内容时被调用。下面创建了一个简单的实例，演示了 Flex 中 CheckBox 组件的具体应用，如代码 9.3 所示。

代码 9.3　使用 CheckBox 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
         import mx.controls.Alert;
         //此事件处理程序从购物车添加和删除项目
         private function modify():void
         {
             hItems.text = "";
            if(read.selected == true) {
           hItems.text += read.label + '\n' ;
           }
              if(sing.selected == true) {
           hItems.text += sing.label + '\n';
           }
           if(playbb.selected == true) {
           hItems.text += playbb.label + '\n';
           }
           }
         // 此事件处理程序显示用户选择结果
         private function sendMessage():void
         {
             if(couponCB.selected == true) {
                 Alert.show('You will receive coupons.');
              }
              else {
                 Alert.show('You will not receive any coupons.');
              }
         }
      ]]>
   </mx:Script>
```

```
    <mx:Panel title="CheckBox Control Example"
        height="75%" width="75%" layout="horizontal"
        paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10">
        <mx:VBox>
            <mx:CheckBox id="read" label="读书" click="modify()"/>
            <mx:CheckBox id="sing" label="唱歌" click="modify()"/>
            <mx:CheckBox id="playbb" label="篮球" click="modify()"/>
        </mx:VBox>
        <mx:VBox>
            <mx:Label text="你的爱好有: "/>
            <mx:TextArea id="hItems" width="300" height="50" verticalScroll
            Policy="off"/>
            <!-- Event handler sendMessages() is used to handle event click -->
            <mx:CheckBox id="couponCB" label="Send me coupons for items "
                click="sendMessage()" selected="false" color="blue"/>
        </mx:VBox>
    </mx:Panel>
</mx:Application>
```

在代码 9.3 中，提供了 3 个 CheckBox 组件，供用户选择自己的爱好，并且设置 CheckBox 组件的 click 事件调用函数 modify()，输出用户选择的结果。具体的效果如图 9-5 所示。

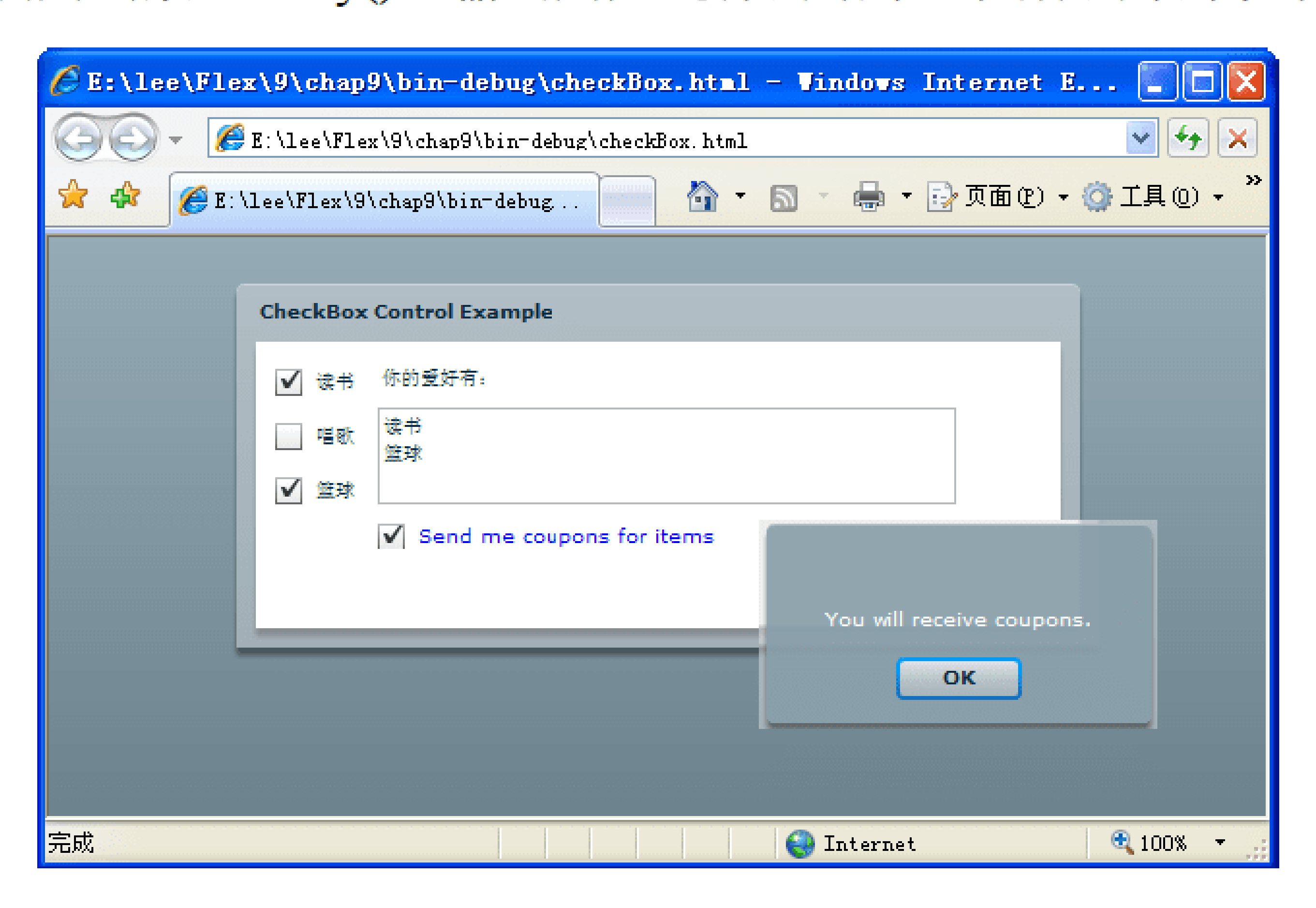


图 9-5　使用 CheckBox 组件

2. RadioButton 组件

RadioButton 组件有一些常用的属性，可以得到 RadioButton 文本、是否选中等信息，该组件的一些属性和 CheckBox 组件相同，这里就不再说明。除此之外，RadioButton 组件有一个特殊的属性 GroupName，该属性获取或者设置单选按钮所属的组名。

在实际的开发应用中，可以向页面添加单个 RadioButton 组件，并单独使用这个组件。但是单选按钮很少单独使用，而是进行分组以提供一组互斥的选项。在一个组内，每次只能选择一个单选按钮。

在 Flex 中，用户还可以使用 RadioButtonGroup 组件创建一组单选按钮，其功能和多个 RadioButton 按钮完全相似。下面就创建使用 RadioButton 组件和 RadioButtonGroup 组件的实例，如代码 9.4 所示。

代码 9.4　使用 RadioButton 和 RadioButtonGroup 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
    <![CDATA[
      import mx.controls.Alert;
      private function show():void
      {
      if(option4.selected)
      {
        if(radiogroup1.selectedValue=="51" ){
        Alert.show("回答正确");
        }
        else
        Alert.show("第二道题出错了！ ")
      }
      else{
      if(radiogroup1.selectedValue=="51"){
      Alert.show("第一道题出错了！ ");
      }
      else
      Alert.show("全错，该补习补习知识了");
      }
      }
    ]]>
   </mx:Script>
   <mx:Panel title="RadioButton Control Example" height="359" width="440"
      paddingTop="10" paddingLeft="10" paddingRight="10">
     <mx:Label width="100%" color="blue" text="2008 年北京奥运会是奥运会历史上
     第几届奥运会？ " fontSize="12"/>
      <mx:RadioButton groupName="num" id="option1" label="26"/>
      <mx:RadioButton groupName="num" id="option2" label="27"/>
      <mx:RadioButton groupName="num" id="option3" label="28"/>
      <mx:RadioButton groupName="num" id="option4" label="29"/>
      <mx:Label text="在北京奥运会上，中国奥运健儿共获得了几枚金牌？ " width="286"
      color="#0622F9" fontSize="12"/>
      <mx:RadioButtonGroup id="radiogroup1"/>
      <mx:RadioButton label="50" groupName="radiogroup1"/>
      <mx:RadioButton label="51" groupName="radiogroup1"/>
      <mx:RadioButton label="52" groupName="radiogroup1"/>
```

```
        <mx:RadioButton label="53" groupName="radiogroup1"/>
        <mx:Button label="提交答案" click="show()"/>
    </mx:Panel>
</mx:Application>
```


在代码 9.4 中，分别使用 RadioButton 组件和 RadioButtonGroup 组件，并在处理函数 show()中使用流程控制语句对用户选择结果进行判断并处理，具体效果如图 9-6 所示。

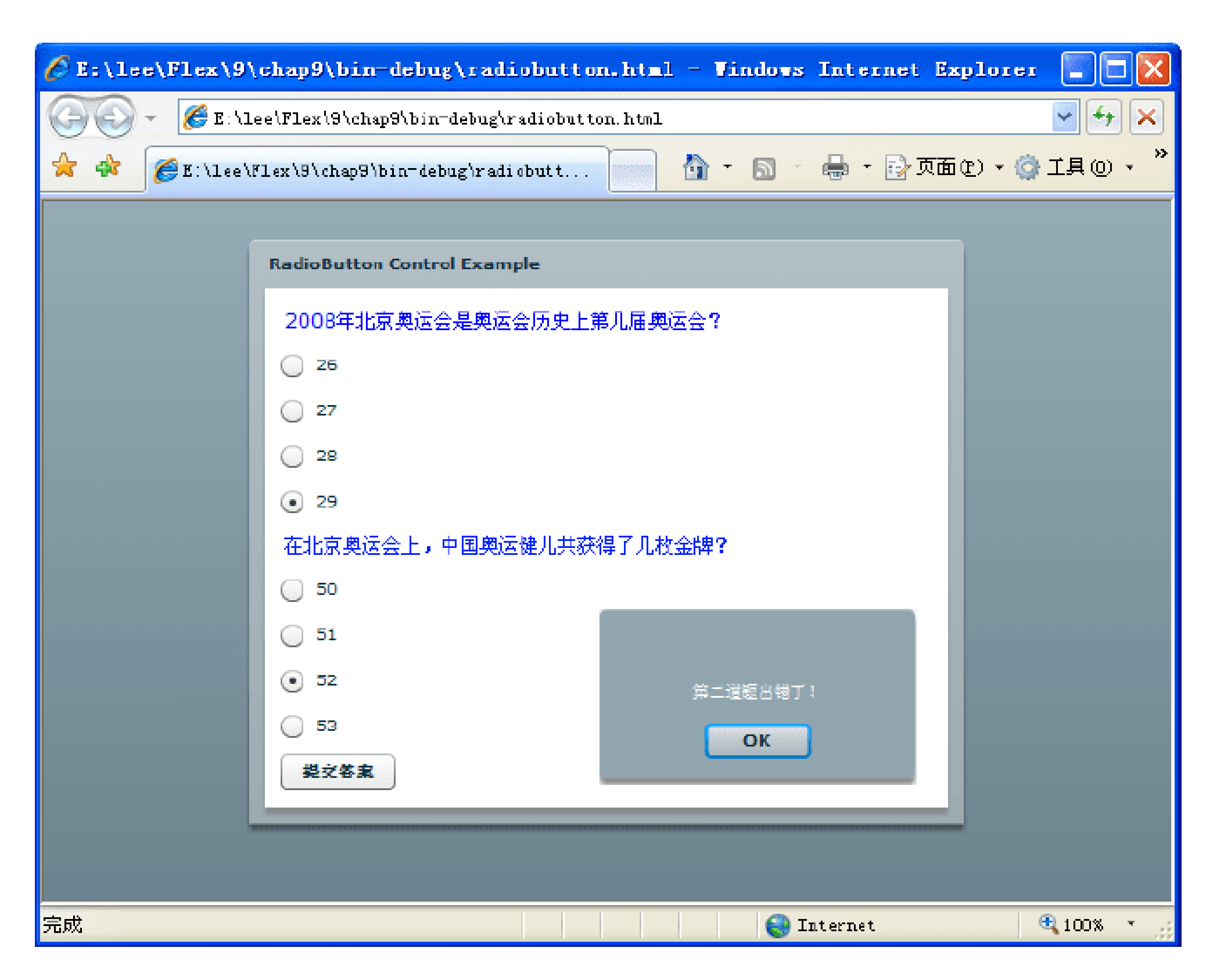


图 9-6 使用 RadioButton 和 RadioButtonGroup 组件

9.2.3 ComboBox 和 List 组件

不管是在网页上，还是在桌面应用程序中，随处可见 ComboBox 和 List 组件。这两个组件的功能很相似，都用于提供选择界面，只是表现形式略有差异。ComboBox 组件是弹出式的下拉列表，List 则把数据元素直接显示出来。

1. ComboBox 组件

使用 ComboBox 组件最重要的一个属性就是 dataprovider。该属性定义了 ComboBox 组件的下拉列表。在实际应用中，通常会定义一个数组，来存放下拉菜单列表项，然后设置该 ComboBox 组件的 dataprovider 属性为该数组，获取具体内容。

使用 ComboBox 时，还需要重点理解 DropdownEvent 事件。在列表被弹出或者收回时，会分别派发 DropdownEvent 对象的 open 事件和 close 事件。

下面创建了一个使用 ComboBox 组件的简单实例。实例插入了一个 ComboBox 组件和几个 Label 组件，再显示用户选择的结果，如代码 9.5 所示。

代码 9.5 使用 ComboBox 组件

```
<?xml version="1.0"?>
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            [Bindable]
            public var seasons: Array = [ {label:"春天", data:1},
                {label:"夏天", data:2}, {label:"秋天", data:3} , {label:"冬天",
                data:4} ];
            [Bindable]
            public var selectedItem:Object;
        ]]>
    </mx:Script>
    <mx:Panel title="ComboBox Control Example"
        height="173" width="75%" layout="horizontal"
        paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10">
        <mx:ComboBox dataProvider="{seasons}" width="150"
            close="selectedItem=ComboBox(event.target).selectedItem"/>
        <mx:VBox width="250">
            <mx:Text  width="200" color="blue" text="选择你喜欢的季节" fontSize=
            "12"/>
            <mx:Label text="You selected: {selectedItem.label}"/>
            <mx:Label text="Data: {selectedItem.data}"/>
        </mx:VBox>
    </mx:Panel>
</mx:Application>
```

在代码 9.5 中，首先声明了变量 selectedItem，然后设置 close 事件，获取用户的选择内容，并在 Label 组件上显示出来。具体效果如图 9-7 所示。

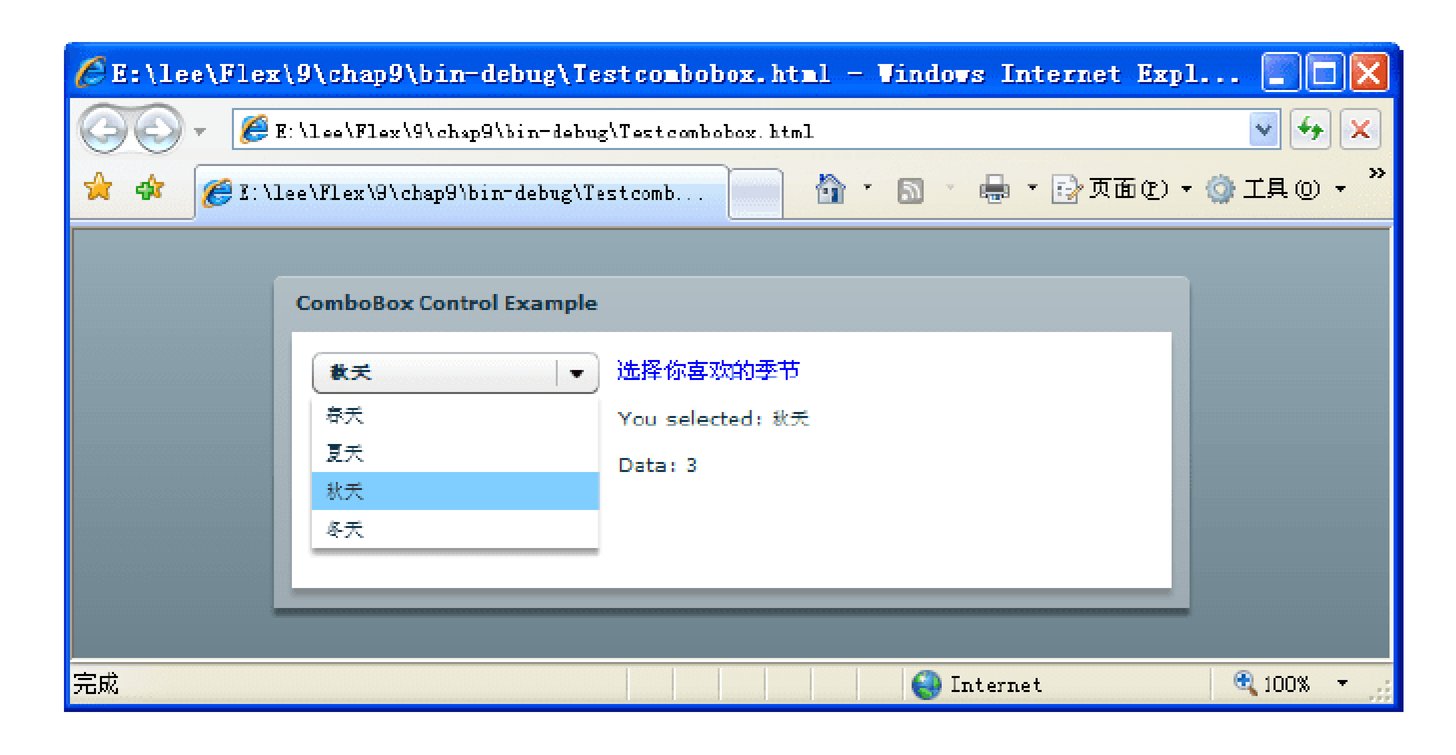


图 9-7　使用 ComboBox 组件

2. List 组件

List 组件和 ComboBox 组件功能相似，但具有本质区别。事实上，List 组件的功能更强大，结构也更复杂。

在获得数据源后，List 组件会分析数据，将每条数据以默认的形式展现出来，这种用来控

制数据表现形式的机制称之为“itemRenderer”。用户可以定义自己的 itemRenderer 对象，覆盖默认的属性。自定义的 itemRenderer 相当于一个简单的自定义组件，同样，里面可以放置其他组件来实现更加丰富、更加灵活的表现形式。所有 itemRenderer 组件都拥有一个 data 属性，每一条数据都对应一个 itemRenderer 实例对象，itemRenderer 对象通过 data 属性来读取数据。

下面创建一个使用 List 组件的简单实例，如代码 9.6 所示。

代码 9.6　使用 List 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
        [Bindable]
           public var selectedItem:Object;
     ]]>
   </mx:Script>
   <mx:Model id="mystates">
     <states>
       <state label="北京" data="京"/>
       <state label="河南" data="豫"/>
       <state label="上海" data="沪"/>
       <state label="重庆" data="渝"/>
       <state label="山东" data="鲁"/>
       <state label="广东" data="粤"/>
       <state label="天津" data="津"/>
     </states>
   </mx:Model>
   <mx:Panel title="List Control Example" height="75%" width="75%"
      paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10">
      <mx:Label text="选择一个省，并查看它的简称"/>
      <mx:List width="370" color="blue"
           dataProvider="{mystates.state}"
           change="this.selectedItem=List(event.target).selectedItem"/>
      <mx:VBox width="100%">
         <mx:Label text="选择省份: {selectedItem.label}"/>
         <mx:Label text="省份简称: {selectedItem.data}"/>
      </mx:VBox>
   </mx:Panel>
</mx:Application>
```

在这个程序中，出现了一个新的组件：Model。Model 组件主要用于定义数据。Model 定义的数据经过编译被转化为一般的 ActionScript 数据对象，可以用作数据绑定。这些数据不可以被更改，没有明确的数据类型，可以是字符串、整形、XML 数据等。

这里，Model 标签里的数据被解析成 ActionScript 对象，被用作 List 组件的数据源。mystates.state 包括了 XML 数据中所有节点为 state 的数据，并通过 dataProvider 属性将数据传

递给 List 组件。

运行该程序，具体效果如图 9-8 所示。

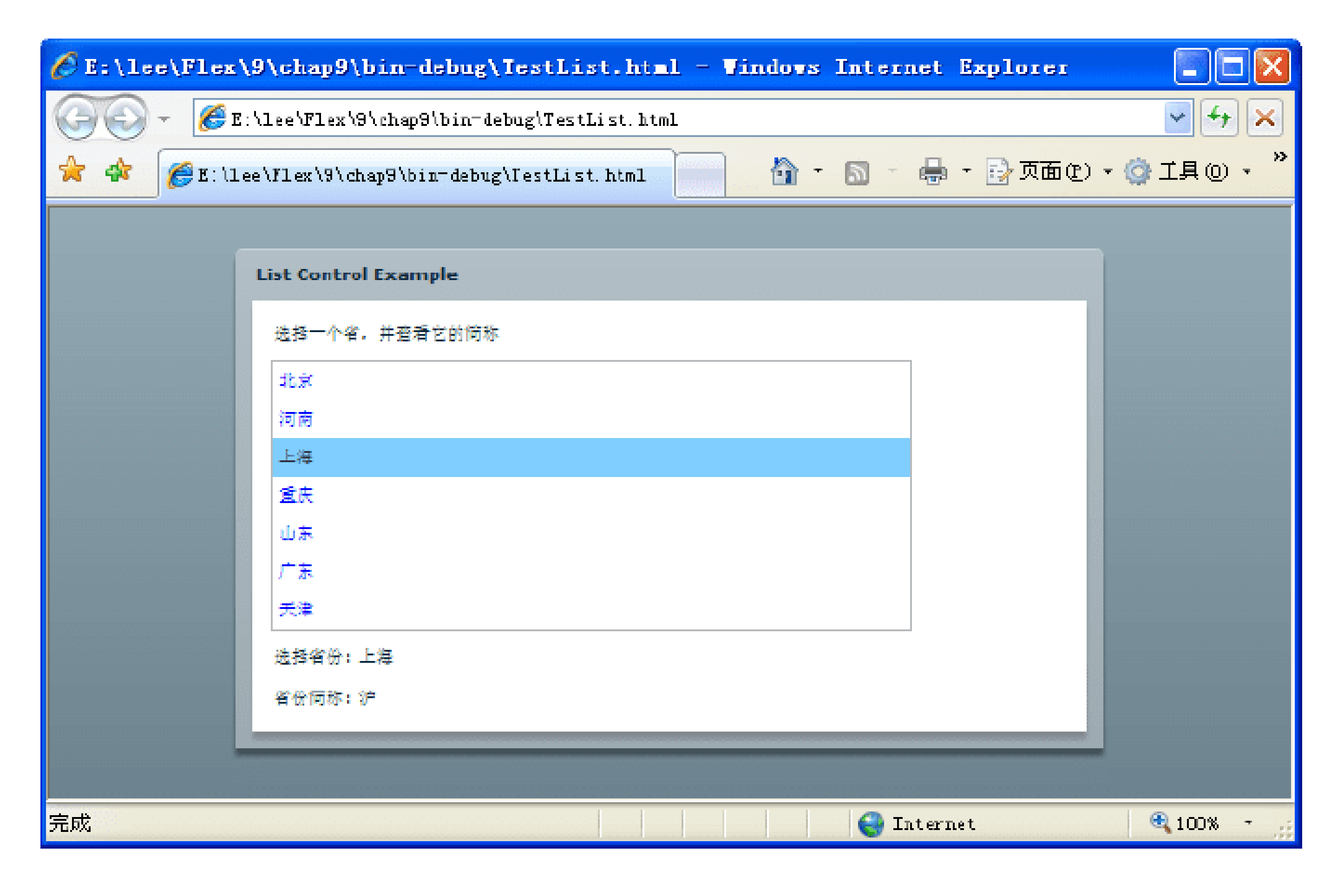


图 9-8　使用 List 组件

9.2.4　按钮组件

在程序中，按钮无疑是使用最频繁的元素之一。它的作用非常明确，就是响应用户的鼠标行为，触发预定义的事件。

1. Button 组件

Button 组件是一个可调整大小的矩形用户界面按钮。在页面中可以通过设置 enabled 属性启用或者禁用按钮。在禁用状态下，按钮不接收鼠标或键盘输入。如果单击或者切换到某个按钮，处于启用状态的 Button 组件就会接收焦点。

在 Flex Builder 3 中为 Button 组件提供了一个 icon 属性，通过该属性可以为按钮设置显示图标，并且可以通过 labelPlacement 属性定义按钮显示文本的位置，该属性有 4 个参数，left、right、top、bottom，分别代表左、右、上、下 4 种显示效果。例如下面代码：

```
<mx:Button x="53" y="167" label="确认" labelPlacement="right" icon="@Embed
(source='../398.gif')" fontSize="16" width="93" height="30">
```

2. LinkButton 组件

LinkButton 组件与 Button 组件功能相似，表 9-3 中列出了 LinkButton 组件一些常用的属性。

表 9-3　LinkButton 组件的常用属性

属性	说明
id	唯一标识 LinkButton 组件
label	LinkButton 组件的显示文本

续表

属性	说明
color	定义文本的显示颜色
themeColor	定义鼠标移动到 Linkbutton 组件上时背景色
alpha	定义鼠标移动到 Linkbutton 组件上时背景图层的透明度
selected	当前组件是否处于选中状态
enable	当前组件是否可用
visible	当前组件是否可见

3．按钮组件应用实例

下面创建一个使用 Button 组件和 LinkButton 组件的实例，演示这两种组件的属性及事件的应用，如代码 9.7 所示。

代码 9.7　使用 Button 和 LinkButton 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
         import flash.events.Event;
        // 事件处理程序打印一条消息显示用户选择了那个按钮
         private function printMessage(event:Event):void {
           message.text += event.target.label + " pressed" + "\n";
         }
     ]]>
   </mx:Script>
   <mx:Panel title="Button Control Example"
      height="75%" width="324" layout="horizontal"
      paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10">
      <mx:VBox height="154">
         <mx:Label width="100%" color="blue"
            text="选择一个按钮" fontSize="12"/>
          <mx:Button id="iconButton" icon="@Embed(source='../398.gif')"
          label="带图标按钮"
             labelPlacement="right" color="#993300" click="printMessage
             (event);" width="117" height="30"/>
          <mx:Button label="定制按钮" color="#993300" toggle="true" selected=
          "true"
             textAlign="left" fontStyle="italic" fontSize="13" width="{icon
             Button.width}"
             click="printMessage(event);"/>
          <mx:Button label="默认按钮" click="printMessage(event);"/>
          <mx:LinkButton label="LinkButton" click="printMessage(event);"
          color="#16D07B" themeColor="#9A09B9"/>
```

```
            </mx:VBox>
             <mx:TextArea id="message" text="" editable="false" height="100%"
             width="100%"
                 color="#0000FF"/>
        </mx:Panel>
    </mx:Application>
```


在代码 9.7 中，创建了 4 个按钮，分别展示了 Button 组件和 LinkButton 组件的各种属性的应用，并定义 click 事件的触发函数。具体的效果如图 9-9 所示。

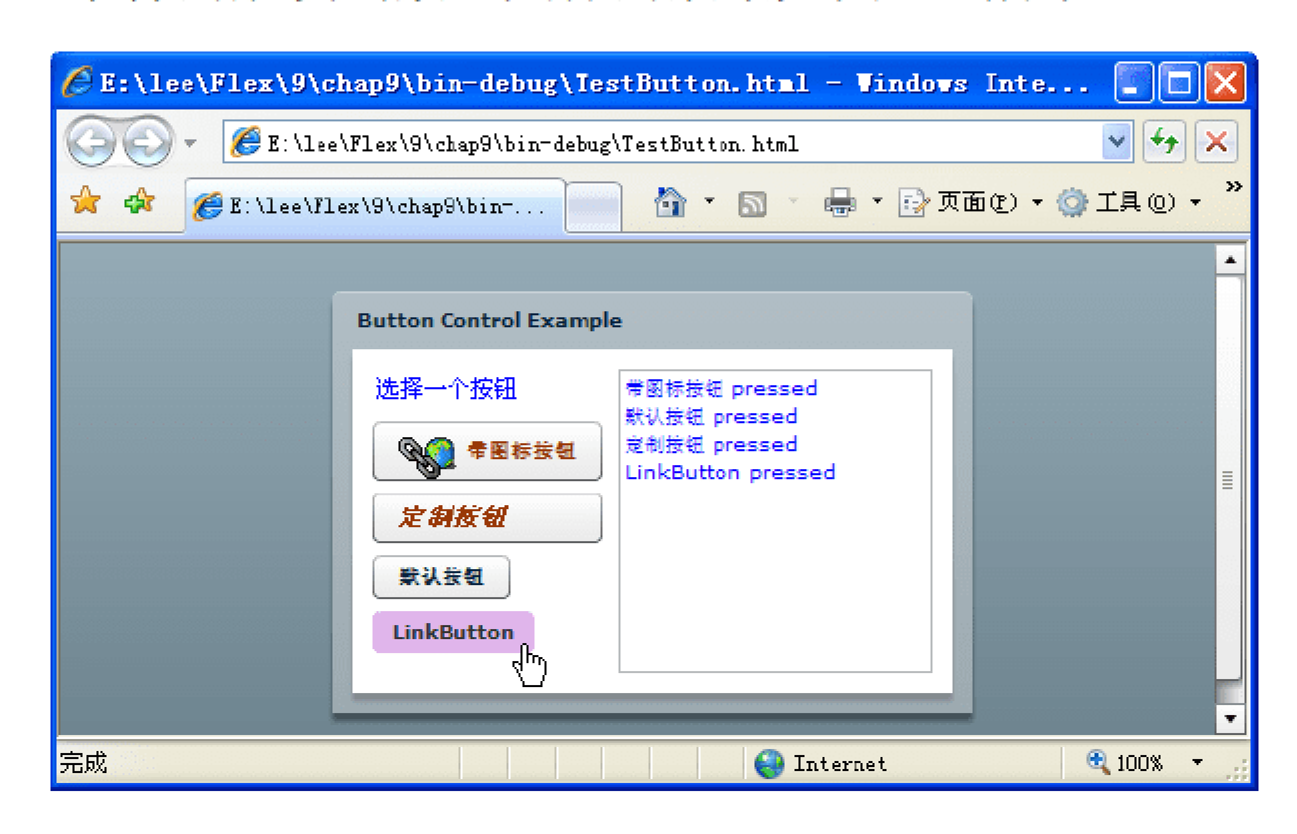


图 9-9　使用按钮组件

9.2.5　Image 组件

Image 组件用于在网页或者桌面应用程序上显示图像，程序员可以添加 Image 组件并指定显示的图片。该组件同样支持数据绑定，允许动态设置其属性值。

Image 组件最常用的属性有两个，id 属性和 source 属性。id 唯一标识一个 Image 组件，source 属性表示该 Image 组件要显示图像的位置。

Image 组件最常用的事件有 mouseMove 事件、mouseOut 事件、mouseDown 事件、mouseUp 事件和 mouseOver 事件等。其中 mouseMove 事件在鼠标移动到图像上时被触发，mouseOut 事件在鼠标离开图片时被触发，mouseOver 事件在鼠标悬停在图像上时被触发，mouseDown 事件在鼠标按下时被触发，mouseUp 事件在鼠标按下并释放时被触发。

下面创建一个简单的实例，使用 Image 组件显示一幅图片，并定义相应的事件处理，如代码 9.8 所示。

代码 9.8　使用 Image 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Panel id="myPanel" title="Image Control Example"
        height="75%" width="75%" horizontalAlign="center"
        paddingTop="10" paddingLeft="10">
```

```
        <mx:Label color="blue" text="在应用程序中显示图片" width="153"/>
        <mx:Image source="../1.jpg" mouseOut="img1.source='../1.jpg'" mouse
        Move="img1.source='../pc.jpg'" id="img1"/>
    </mx:Panel>
</mx:Application>
```

代码 9.8 中定义了两个事件，mouseMove 事件和 mouseOut 事件。图 9-10 就为鼠标移动到图像时的显示效果。

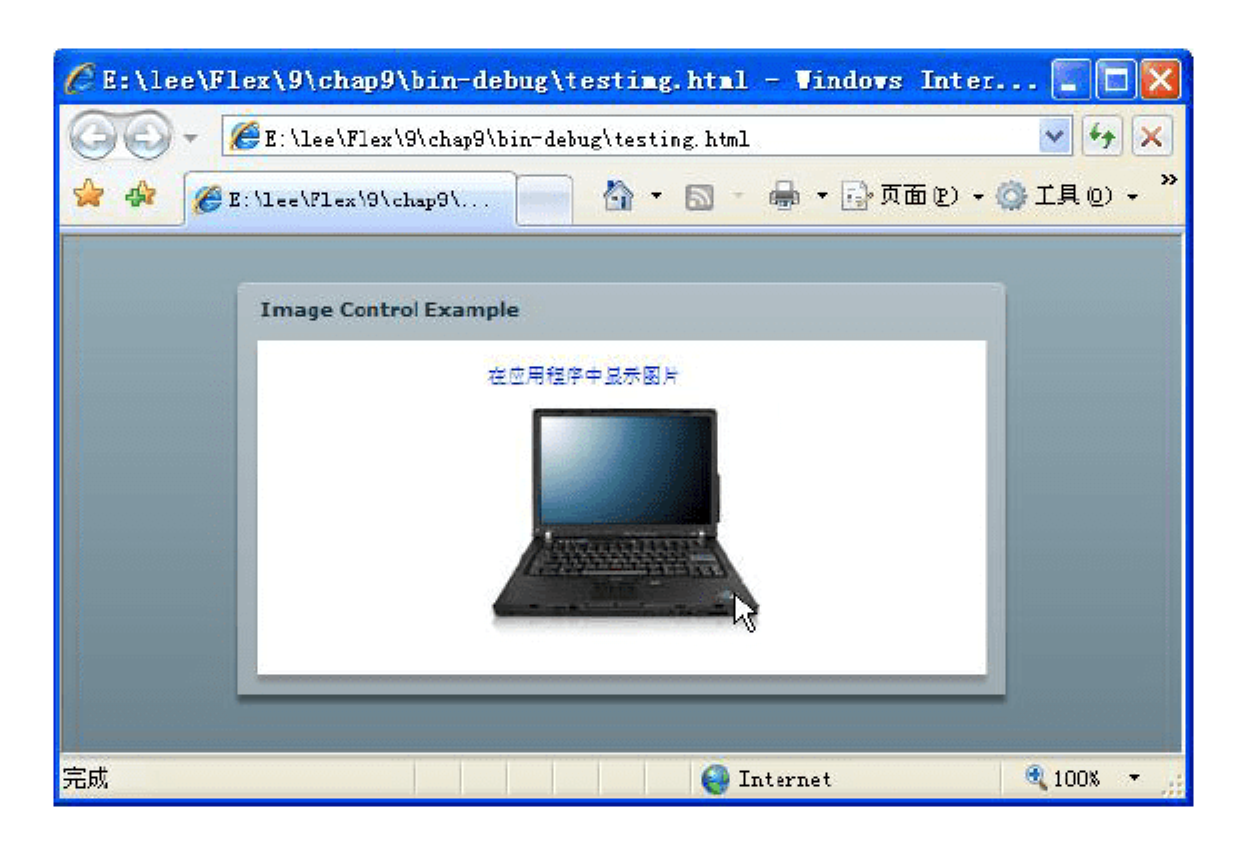


图 9-10　使用 Image 组件

9.2.6　日期组件

在 Flex Builder 3 中，提供了两种日期组件：DateChooser 组件和 DateField 组件。本小节将介绍这两种组件的具体应用。

1. DateChooser 组件

DateChooser 组件显示了一个日历列表，通过单击相应的按钮，用户可以选择任意一个日期。DateChooser 组件常用的属性如表 9-4 所示。

表 9-4　DateChooser 组件的常用属性

属性	说明
id	唯一标识 DateChooser 组件
displayedYear	默认加载时显示的年份
displayedMonth	默认加载时显示的月份
showToday	当前日期是否突出显示
selectedDate	默认选择日期
visible	当前 DateChooser 组件是否可见
disabledRanges	禁用某一天或一个范围内的日期

DateChooser 组件最常用的一个事件就是 change 事件，当用户选择的日期发生改变时触发该事件，执行具体的处理函数。

在程序中还使用 DateFormatter 对象定义日期显示的格式。DateFormatter 对象的方法和属性如下所示。

- **方法 format(value:Object):String**　可以是 Date 类型或者是一个 Date 类型格式的 String 类型。
- **属性 formatString**　例如：formatString="YYYY-MM-DD"
- **保留字 Y|M|D|A|E|H|J|K|L|N|S**　保留字的具体应用如表 9-5 所示。

表 9-5　formatString 保留字格式及说明

保留字	说明
Y	年，以 2009 年为例：YY=09 YYY=009 YYYY=2009 少于两个按两个计算
M	月，以 7 月为例：M=7 MM=07 MMM=Jul(英文简写) MMMM=July(英文全称)
D	日，以 1 日为例：D=1 DD=01
E	星期，以星期一为例：E=1 EE=01 EEE=Mon EEEE=Monday
A	上下午，AM/PM
J	24 小时制，结果为 0～23
H	24 小时制，结果为 1～24
K	12 小时制，结果为 0～11
L	12 小时制，结果为 1～12
N	分钟，以 3 分钟为例：N=3 NN=03
S	秒，以 3 秒为例：SS=03

例如：

```
EEEE, MMM. D, YYYY at H:NN A = Tuesday, Sept. 8, 2005 at 1:26 PM
YYYY-MM-DD HH:NN:SS=2009-1-27 15:15:15
```

下面创建了一个使用 DateChooser 组件的实例，如代码 9.9 所示。

代码 9.9　使用 DateChooser 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            // 事件处理程序显示用户选择的日期
            private function displayDate(date:Date):void {
                if (date == null)
                    selection.text = "Date selected: ";
                else
                    selection.text = "Date selected: " + date.getFullYear().to
                    String() +
                        '/' + (date.getMonth()+1).toString() + '/' + date.get
                        Date();
            }
        ]]>
    </mx:Script>
```

```
        <mx:DateFormatter id="df" formatString="MMMM/DD YYYY" />
        <mx:Panel title="DateChooser Control Example" height="75%" width="75%"
            paddingTop="10" paddingLeft="10" paddingRight="10">
            <mx:Label width="100%" color="blue"
                text="Select a date in the DateChooser control. Select it again to
                clear it."/>
            <mx:HBox horizontalGap="25">
              <mx:VBox>
                  <mx:Label text="Simple DateChooser control."/>
                  <mx:DateChooser id="dateChooser1" yearNavigationEnabled="true"
                      change="displayDate(DateChooser(event.target).selected
                      Date)"/>
                  <mx:Label id="selection"  color="blue" text="Date selected:"/>
              </mx:VBox>
              <mx:VBox>
                  <mx:Label text="Disable dates before April 10, 2006."/>
                  <mx:DateChooser id="dateChooser2" yearNavigationEnabled="true"
                      disabledRanges="{[ {rangeEnd: new Date(2006, 3, 10)} ]}"/>
                <mx:Label  color="blue" text="Date selected: {df.format(date
                Chooser2.selectedDate)}"/>
              </mx:VBox>
            </mx:HBox>
        </mx:Panel>
    </mx:Application>
```

在代码 9.9 中，使用了两个 DateChooser 组件。第一个组件设置其 change 事件调用函数 displayDate()，并将用户选择的日期作为参数传递过去。在处理函数中，获取用户选择日期的年、月、日，并按照特定的形式显示出来。第二个组件设置其 disabledRanges 属性，使得 2006 年 4 月 10 号之前的日期被禁用。

在浏览器中运行，具体的效果如图 9-11 所示。

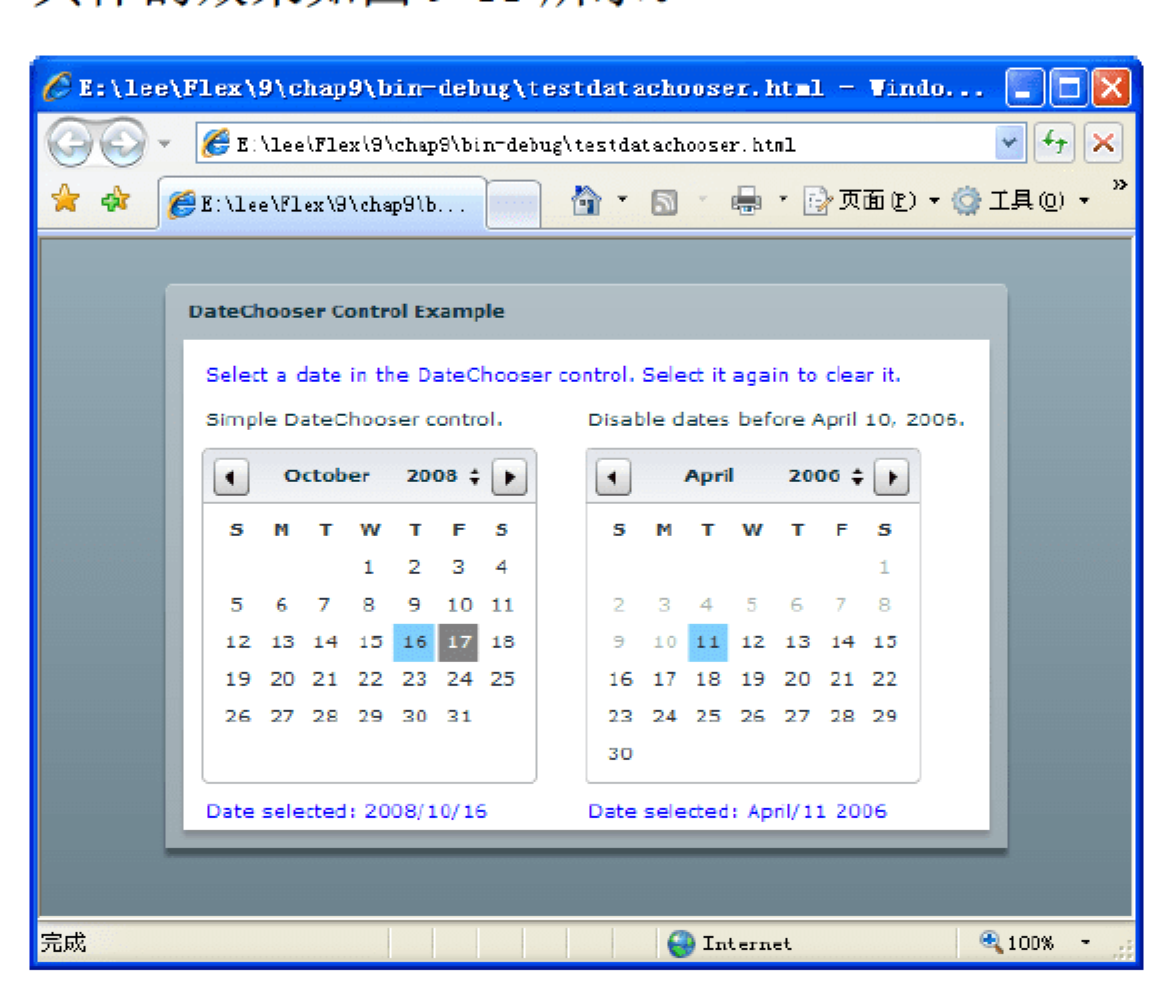


图 9-11　使用 DataChooser 组件

2. DateField 组件

DateField 组件用于接受用户选择的日期信息，显示为两部分：一个下拉列表（带有以文本形式表示的日期）和一个网格日历列表（在单击列表时显示）。

DateField 组件允许用户在许多不同的格式中选择一个日期值，可以以任何标准日期格式显示基于 Data 的值。在用户单击时，会显示一个日历列表，允许用户选择日历中的一个日期。

DateField 组件的属性与 DateChooser 组件的属性相似，这里就不详细介绍。除了前面介绍过的属性外，DateField 还包含一些属性，这些属性可以改变日期的外观、显示字体等。

下面举例说明 DateField 组件如何使用，在程序中使用 DateField 组件提示用户选择日期，并在 Label 组件中显示出来，如代码 9.10 所示。

代码 9.10　使用 DateFiled 组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
     <![CDATA[
        // DateFiled 组件日期改变时的事件处理程序
        private function dateChanged(date:Date):void {
           if (date == null)
              selection.text = "Date selected: ";
           else
             selection.text = "Date selected: " + date.getFullYear().toString() +
                 '/' + (date.getMonth()+1).toString() + '/' + date.getDate();
        }
     ]]>
   </mx:Script>
 <mx:DateFormatter id="df"/>
   <mx:Panel title="DateField Control Example" height="75%" width="75%"
      paddingTop="10" paddingLeft="10" paddingRight="10">
      <mx:Label width="100%"  color="blue"
         text="Select a date in the DateField control. Select it again to clear
         it."/>
      <mx:Label text="Basic DateField:"/>
      <mx:DateField id="dateField1" yearNavigationEnabled="true"
         change="dateChanged(DateField(event.target).selectedDate)" />
      <mx:Label id="selection"  color="blue" text="Date selected:" />
      <mx:Label text="Disable dates before April 10, 2006."/>
      <mx:DateField id="dateField2" yearNavigationEnabled="true"
         disabledRanges="{[ {rangeEnd: new Date(2006, 3, 10)} ]}" />
      <mx:Label  color="blue" text="Date selected: {df.format(dateField2.
      selectedDate)}"/>
   </mx:Panel>
</mx:Application>
```

在代码 9.10 中，使用了两个 DateField 组件，并且使用 DateFormatter 对象定义日期显示格式，具体的效果如图 9-12 所示。

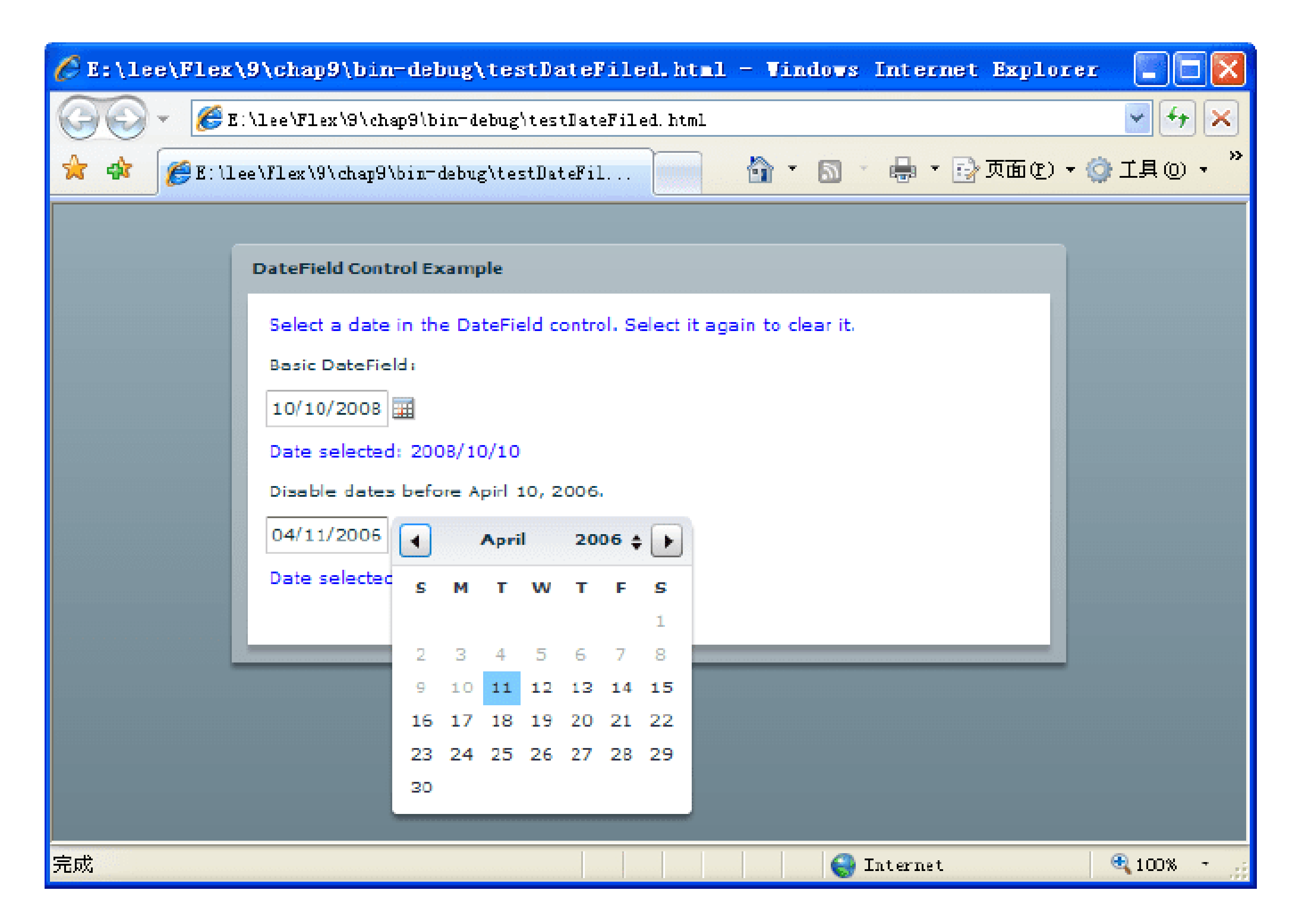


图 9-12　使用 DateField 组件

9.3　导航类组件

前面介绍了 Flex 中的一些常用组件，虽然能实现很多网页的界面效果，但对于像树状结构图、导航菜单等效果实现时仍然比较复杂，而且用户需要熟练掌握脚本语言。在 Flex Builder 3 中提供了实现这些功能的组件，这样就简化了很多复杂功能的实现过程。

9.3.1　ToggleButtonBar 和 TabBar 组件

ToggleButtonBar 是 ButtonBar 的子类，它增强了导航功能，可以保持客户端状态，同时在界面上对当前的选中状态做明确标识。TabBar 在 ToggleButtonBar 的基础上扩展，改变了对选中状态的表现方式。

1. ToggleButtonBar 组件

ToggleButtonBar 组件定义一组水平或垂直按钮，这些按钮保持选中或者取消选中状态。在 ToggleButtonBar 组件中，只有一个按钮可以处于选中状态。这就意味着，当用户从 ToggleButtonBar 组件中选择一个按钮后，该按钮将保持选中状态直到用户选择其他按钮。

如果将 ToggleButtonBar 组件的 toggleOnClick 属性设置为 true，则当前选中的按钮将被取消其选中状态。默认情况下，toggleOnClick 属性设置为 false。

使用 ToggleButtonBar 组件的 direction 属性，可以设置按钮是水平排列还是垂直排列。通过配置 verticalGap 属性和 horizontalGap 属性可以调整按钮之间的间距。

ToggleButtonBar 组件还有一个重要的属性 selectedIndex，该属性在创建组件时确定选中

哪个按钮。默认值为 0，表示选择栏中最左侧的按钮。将 selectedIndex 属性设置为–1 可以取消对栏中所有按钮的选择。

下面创建一个使用 ToggleButtonBar 组件的实例，如代码 9.11 所示。

代码 9.11　使用 ToggleButtonBar 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
       import mx.events.ItemClickEvent;
       private function clickHandler(event:ItemClickEvent):void {
          myTA.text="被选按钮序号: " + String(event.index) +
             "\n" + "被选择的按钮为: " + event.label;
         }
      ]]>
   </mx:Script>
   <mx:Panel title="ToggleButtonBar Control Example" height="211" width="366"
      paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom="10">
      <mx:Label width="100%" color="blue"
         text="选择一个按钮." fontSize="12"/>
      <mx:HBox>
        <mx:ToggleButtonBar itemClick="clickHandler(event);" width="106"
        themeColor="#F82818" verticalGap="5" direction="vertical">
           <mx:dataProvider>
              <mx:Array>
                 <mx:String>FLEX</mx:String>
                 <mx:String>FLASH</mx:String>
                 <mx:String>DREAMWEAVER</mx:String>
                 <mx:String>PHOTOSHOP</mx:String>
              </mx:Array>
           </mx:dataProvider>
        </mx:ToggleButtonBar>
        <mx:TextArea id="myTA" width="210" height="100%"/>
      </mx:HBox>
   </mx:Panel>
</mx:Application>
```

在代码 9.11 中，创建了 4 个垂直排列的按钮，并设置按钮间的间距为 5 像素，设置选中时按钮边框颜色为#F82818，当用户单击按钮时，在 TextArea 文本区域中显示具体的选择结果。具体效果如图 9-13 所示。

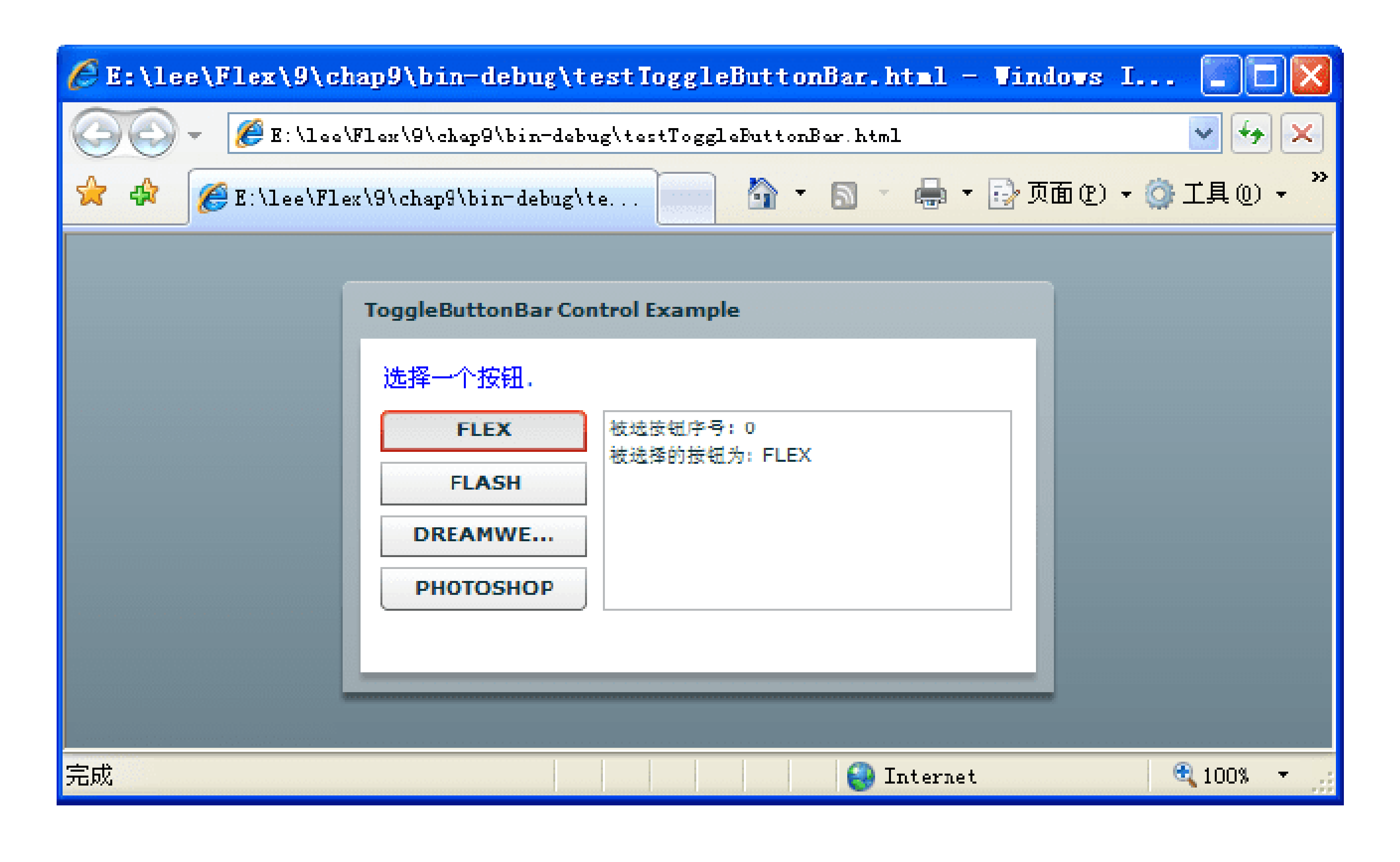


图 9-13　使用 ToggleButtonBar 组件

2. TabBar 组件

TabBar 组件在 ToggleButtonBar 组件的基础上进行了扩展，改变了对选中状态的显示方式，将选中项以标签的形式突出显示。TabBar 组件具有独立性，使得它可以和任何组件结合在一起使用。

TabBar 组件的属性和 ToggleButtonBar 组件的各属性完全相似，这里不再重复。下面创建一个使用 TabBar 组件的实例，如代码 9.12 所示。

代码 9.12　使用 TabBar 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
         import mx.events.ItemClickEvent;
         import mx.controls.TabBar;
         [Bindable]
         public var STATE ARRAY:Array = [{label:"ItZcn", data:"www.itzcn.
         com"},
            {label:"WebZcn", data:"www.webzcn.com"},
            {label:"BAIDU", data:"www.baidu.com"}
         ];
        private function clickEvt(event:ItemClickEvent):void {
           //访问 TabBar 控制目标
           var targetComp:TabBar = TabBar(event.currentTarget);
           forClick.text="网站名称是: " + event.label + ", 序号是: " +
              event.index + ", 网址为: " +
              targetComp.dataProvider[event.index].data;
        }
      ]]>
   </mx:Script>
```

```
        <mx:Panel title="TabBar Control Example" height="188" width="75%"
            paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10">
            <mx:Label width="100%" color="blue"
                text="选择一个标签来改变当前的组." fontSize="12"/>
            <mx:TabBar itemClick="clickEvt(event);">
                <mx:dataProvider>{STATE_ARRAY}</mx:dataProvider>
            </mx:TabBar>
            <mx:TextArea id="forClick" height="100%" width="100%"/>
        </mx:Panel>
    </mx:Application>
```

在代码 9.12 中，定义了一个 TabBar 组件，并创建了一个数据，存放各个标签信息。监听 TabBar 组件的 itemClick 事件，并在函数 clickEvt()中做出相应处理，显示用户的选择内容。具体效果如图 9-14 所示。

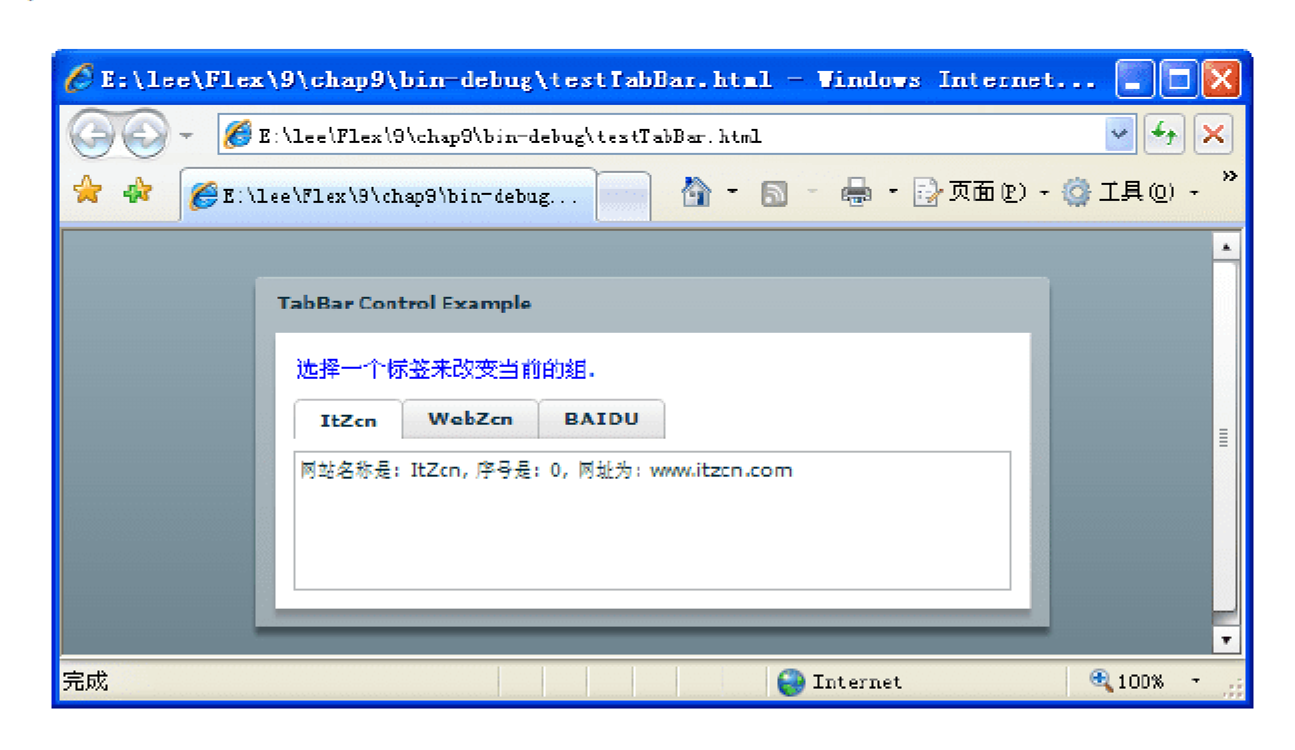


图 9-14　使用 TabBar 组件

9.3.2　MenuBar 组件

在程序中，菜单的用途广泛，例如操作系统中，随处可见各式各样的菜单。菜单具有体积小、使用方便等特点，合理利用了空间，用户操作起来也比较直观，是导航系统的理想选择。

MenuBar 组件最主要的属性就是 dataProvider，该属性定义了菜单的所有内容项。MenuBar 组件和所有的组件一样，支持 XML 数据或者数组作为数据源。

MenuBar 组件还支持多级菜单，下面就创建一个简单的实例，在实例中定义了多级菜单，如代码 9.13 所示。

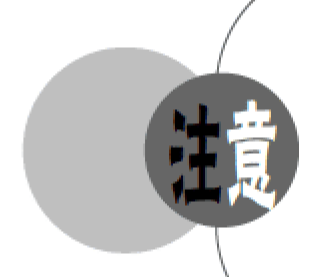
代码 9.13　使用 MenuBar 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete=
"initCollections();" >
    <mx:Script>
        <![CDATA[
```

```
            import mx.events.MenuEvent;
            import mx.controls.Alert;
            import mx.collections.*;
            [Bindable]
            public var menuBarCollection:XMLListCollection;
            private var menubarXML:XMLList =
               <>
                  <menuitem label="文件" data="file">
                     <menuitem label="新建" data="new"/>
                     <menuitem label="保存" data="save"/>
                     <menuitem type="separator" />
                     <menuitem label="退出系统" data="exit"/>
                  </menuitem>
                  <menuitem label="编辑" data="edit">
                     <menuitem label="撤销" type="check"  data="undo"/>
                     <menuitem label="重做" type="check"  data="redo"/>
                     <menuitem type="separator"/>
                     <menuitem label="操作" >
                        <menuitem label="剪切" type="radio"
                           groupName="one" data="cut"/>
                        <menuitem label="复制" type="radio"
                           groupName="one" data="copy"/>
                     </menuitem>
                  </menuitem>
               </>;
            private function initCollections():void {
               menuBarCollection = new XMLListCollection(menubarXML);
            }
            private function menuHandler(event:MenuEvent):void  {
               if (event.item.@data != "exit") {
                  Alert.show("你选择了: " + event.item.@label + "命令\n" +
                     "Data: " + event.item.@data, "结果");
               }
            }
         ]]>
      </mx:Script>
      <mx:Panel title="MenuBar Control Example" height="215" width="379"
         paddingTop="10" paddingLeft="10">
         <mx:Label width="100%" color="blue"
           text="选择菜单命令" fontSize="12"/>
         <mx:MenuBar labelField="@label" itemClick="menuHandler(event);"
            dataProvider="{menuBarCollection}"  fontSize="12" width="200"/>
      </mx:Panel>
   </mx:Application>
```

在代码 9.13 中，使用 XML 数据的 type 属性，可以直接在 XML 数据中控制菜单的分割线和菜单的具体类型。当 type 属性为 separator 时，表示节点左右一条分割线，而不是菜单项。

在程序中，使用@符号解析 XML 数据，获得节点的属性值。其中@label 表示节点的 laber 属性值，@data 表示节点的 data 属性值。MenuBar 组件自动解析 XML 后，生成多级菜单。



请注意区别 XML 和 XMLList 这两个对象。XML 和 XMLList 都用来定义 XML 数据，不同的是，XMLList 是多个 XML 数据的集合。在 XML 语法中，根节点有且只有一个。在上面的实例中使用了 XMLList 类型，根节点被忽略，menuitem 节点被解析为独立的 XML 数据。

运行程序，具体的效果如图 9-15 所示。

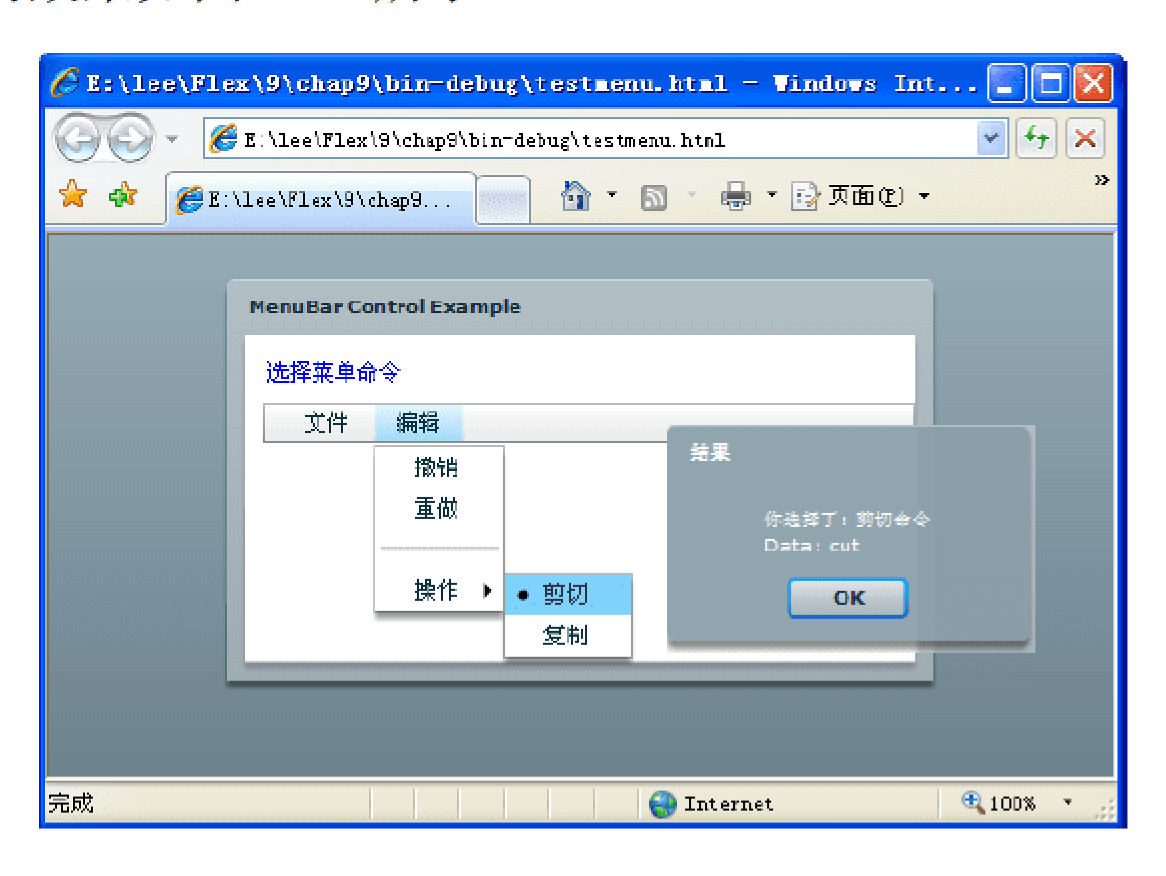


图 9-15　使用菜单导航

9.3.3　PopUpButton 和 PopUpMenuButton 组件

在网站的实际开发过程中，PopUpButton 组件和 PopUpMenuButton 组件会经常用到，当用户单击时，能够弹出相应的菜单选项。

1．PopUpButton 组件

PopUpButton 组件是一个特殊的按钮，它本身有两个按钮组成：主按钮和子按钮。单击主按钮，可以将任何组件作为窗口弹出，置于最上层，这个弹出动作由 PopUpManager 完成。

在 PopUpButton 的事件列表中，比较重要的事件是 open 事件和 close 事件，前者在弹出动作发生时触发，后者在弹出窗口关闭时触发。

PopUpButton 组件的右侧带有箭头标识的按钮为子按钮，也称弹出按钮。单击子按钮，可以打开弹出窗口，和使用 open()方法功能相似。单击主按钮时，所触发的事件与普通 Button 按钮组件是一样的。

下面创建了一个使用 PopUpButton 组件的实例，如代码 9.14 所示。

代码 9.14　使用 PopUpButton 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.controls.*;
            import mx.events.*;
            private var myMenu:Menu;
            private function initMenu():void {
                myMenu = new Menu();
                var dp:Object = [{label:"天气查询"},{label:"信息查询"},{label:
                "图片查询"}];
                myMenu.dataProvider = dp;
                myMenu.selectedIndex = 0;
                myMenu.addEventListener("itemClick", itemClickHandler);
                popB.popUp = myMenu;
                popB.label = "进入: " + myMenu.dataProvider[myMenu.selected
                Index].label;
            }
            private function itemClickHandler(event:MenuEvent):void {
                var label:String = event.item.label;
                popTypeB.text=String("转移到 " + label);
                popB.label = "进入: " + label;
                popB.close();
                myMenu.selectedIndex = event.index;
            }
        ]]>
    </mx:Script>
   <mx:Panel title="PopUpButton Control Example" height="75%" width="75%"
      paddingTop="10" paddingBottom="10" paddingRight="10" paddingLeft="10">
      <mx:Label width="100%" color="blue"
         text="按钮显示文本将包含的最后选定的菜单项." />
    <mx:PopUpButton id="popB" label="弹出菜单" creationComplete="initMenu();"
    width="135" />
         <mx:Spacer height="50" />
      <mx:TextInput id="popTypeB" />
           </mx:Panel>
</mx:Application>
```

在代码 9.14 中，首先使用函数创建了一个 Menu 菜单，为该菜单设置具体选项，并设置监听 itemClick 事件。然后，设置 PopUpButton 组件的 PopUp 属性，显示该 Menu 菜单。当然也可以在函数中使用 open()方法，当页面加载时，自动弹出命令菜单。

接下来声明 itemClick 事件处理函数，当用户选择一项命令时，显示用户选择的内容，并且触发 PopUpButton 组件的 close 事件，关闭菜单。

运行该程序，具体的显示效果如图 9-16 所示。

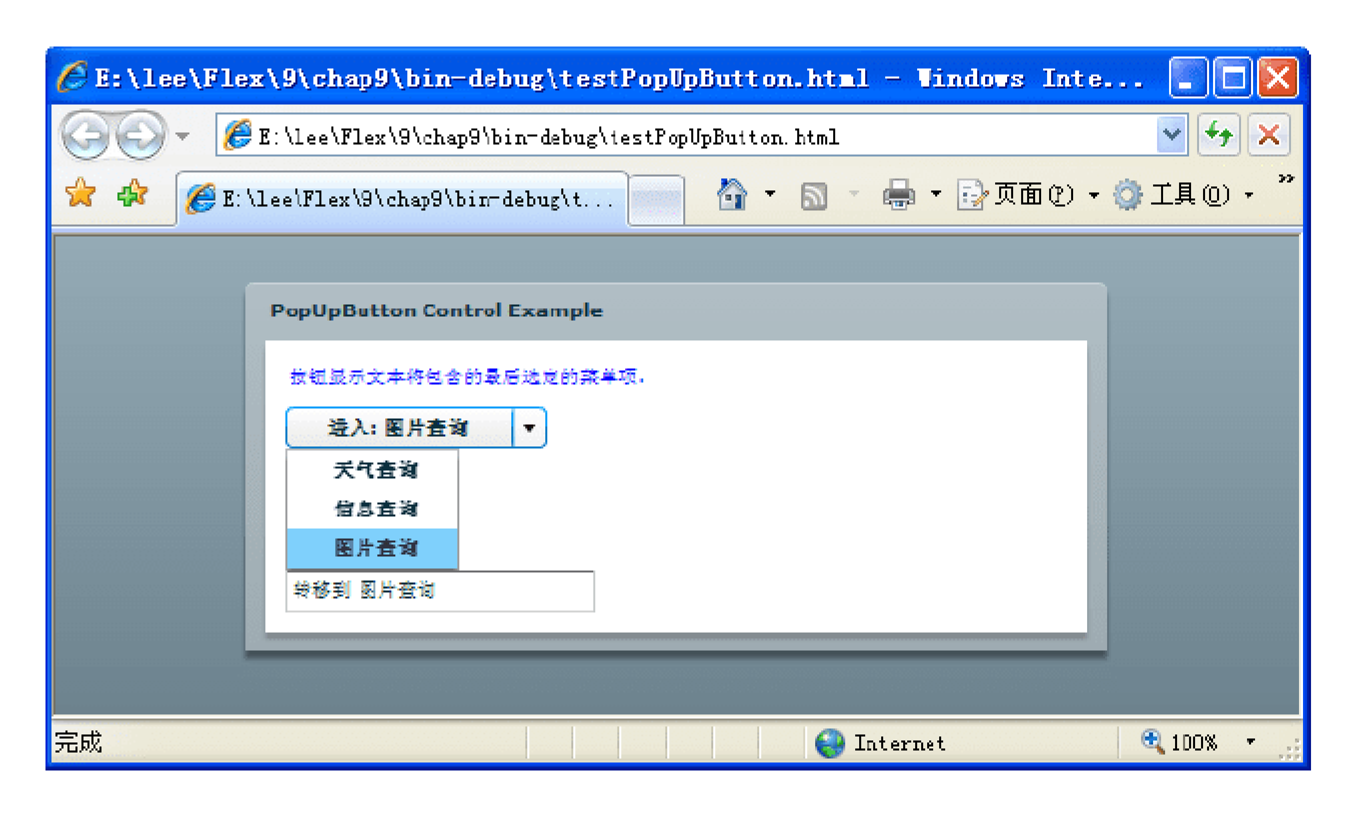
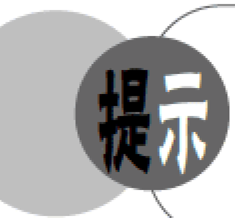


图 9-16　使用 PopUpButton 组件



PopUpButton 组件除了可以弹出菜单外，还可以弹出其他组件，如 Panel、Text Area 等。

2. PopUpMenuButton 组件

PopUpMenuButton 组件继承自 PopUpButton 组件，是 PopUpButton 组件的一个特殊实例。它只能把 Menu 组件当作弹出窗口。

Menu 组件用来创建菜单，不过和 MenuBar 组件相比，Menu 组件缺少了菜单条，而且 Menu 组件没有对应的 MXML 标签，只能由代码创建。

PopUpMenuButton 组件中内置了一个 Menu 组件。下面是一个使用 PopUpMenuButton 组件的实例，如代码 9.15 所示。

代码 9.15　使用 PopUpMenuButton 组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.events.*;
            import mx.controls.*;
            public function itemClickHandler(event:MenuEvent):void {
                Alert.show("选择的菜单命令为: " + event.label
                    + "  \n 选择的菜单命令序号:  " + event.index);
            }
        ]]>
    </mx:Script>
    <mx:XMLList id="treeDP2">
        <node label="工资管理"/>
```

```
        <node label="考勤管理"/>
        <node label="人事管理"/>
        <node type="separator"/>
        <node label="退出系统"/>
    </mx:XMLList>
    <mx:Panel title="PopUpMenuButton Control Example" height="100%" width=
    "100%"
        paddingTop="10" paddingLeft="10" paddingRight="10">
         <mx:Label width="100%" color="blue"
            text="打开菜单" fontSize="12"/>
        <mx:PopUpMenuButton id="p2"
            dataProvider="{treeDP2}"
            labelField="@label"
            itemClick="itemClickHandler(event);" fontSize="12"/>
     </mx:Panel>
</mx:Application>
```

在代码 9.15 中，使用 ActionScript 定义了一串 XML 数据，传给 PopUpMenuButton 组件的 dataprovider 属性，实质上就是作为 PopUpMenuButton 组件内置 Menu 的数据源。此外，代码中，还对 Menu 菜单的 itemClick 事件创建了监听函数，在函数中，使用 Alert.show()方法输出用户的选择结果。

运行该程序，具体效果如图 9-17 所示。

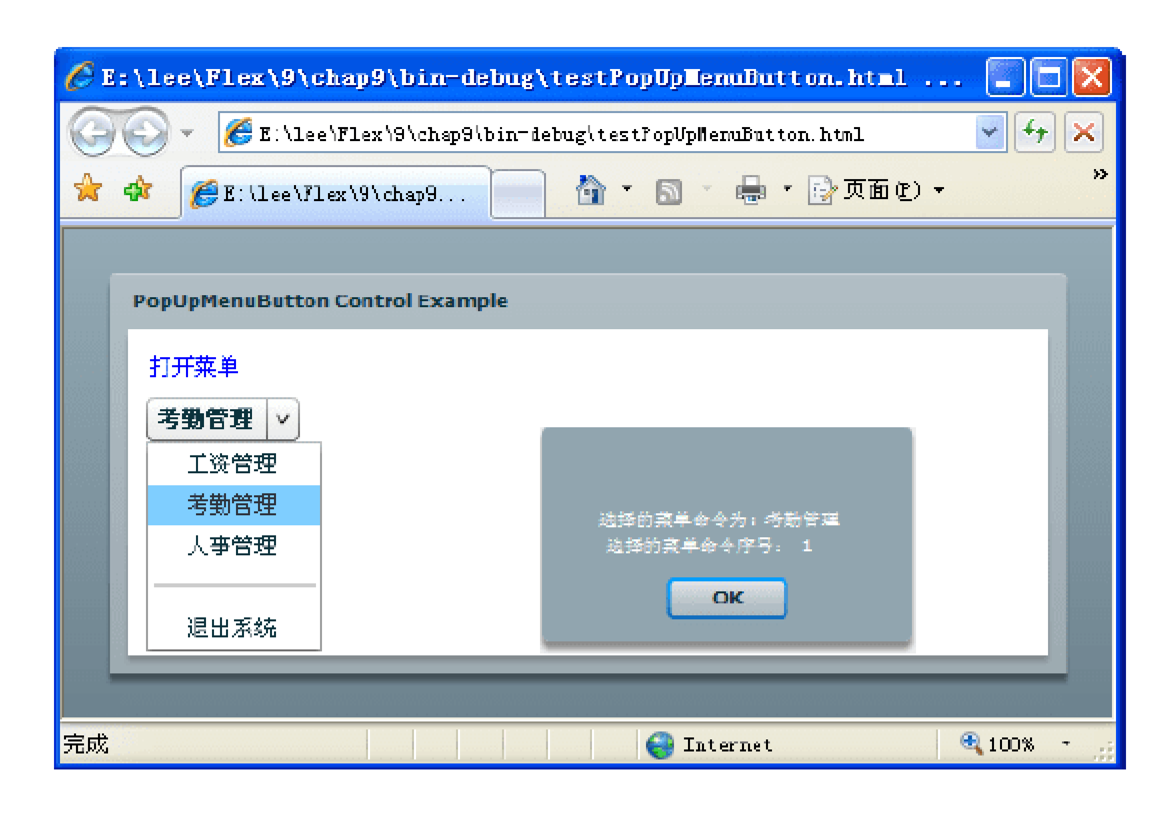


图 9-17　使用 PopUpMenuButton 组件

第10章 使用容器布局页面



内容摘要 | Abstract

在 Flex 中，界面由组件组成，而组件又被放置在一个个特定的组件中，可以把这类组件形象地称为容器。容器类组件都位于 mx.containers 包中。在组件的层级关系中，Container 类是 UIComponent 的子类，是所有 Flex 容器类组件的父类，每个容器组件都在其基础上添加自己的功能。

本章将详细介绍各种 Flex 容器类组件的属性和使用方法，并且介绍如何使用容器对页面进行合理布局。

学习目标 | Objective

- 了解 Application 组件
- 熟悉使用各种与程序布局相关的组件
- 熟悉使用 Panel 组件和 TitleWindow 组件
- 掌握表单组件的用法
- 了解动态控制对象布局的概念
- 熟悉使用各种导航容器的方法

10.1 管理程序的布局

容器是 Flex 的重要特色，正是由于这一点使得 Flex 在网络应用中的效率得到显著提高。因此，界面控制成为程序开发中非常重要的一环，直接体现了程序的质量。下面将详细介绍如何使用组件管理程序的布局。

10.1.1 控制 Application 组件的布局

Application 组件是一个特殊的容器，位于界面元素层级的根部，包含了整个应用程序中的所有元素。在 Application 组件的相关属性中，有 3 个属性与布局息息相关，分别是：layout、horizontalAlign 和 verticalAlign。这 3 个属性存在着依附关系，其中 layout 属性起决定作用，另外两个属性受制于它。

layout 属性有 3 个可选值：absolute、vertical、horizontal。

当 layout 属性值为 absolute 时，horizontalAlign 和 verticalAlign 属性将不起作用，界面上的元素将由各自的坐标来定位。如果元素的 x、y 坐标值都为空，那么就默认为 0。当 layout 属性值为 vertical 或者 horizontal 时，元素的位置由 horizontalAlign 和 verticalAlign 这两个属性来控制。

除了前面介绍的这 3 个属性之外，还可以通过设置 Application 的 width 和 height 属性，设置应用程序界面的宽度和高度，控制最后编译生成的.swf 文件的尺寸。

在设计视图中，单击空白程序，就可以选择 Application 组件。在属性面板中，就可以看到 Application 组件的常见属性，如图 10-1 所示。

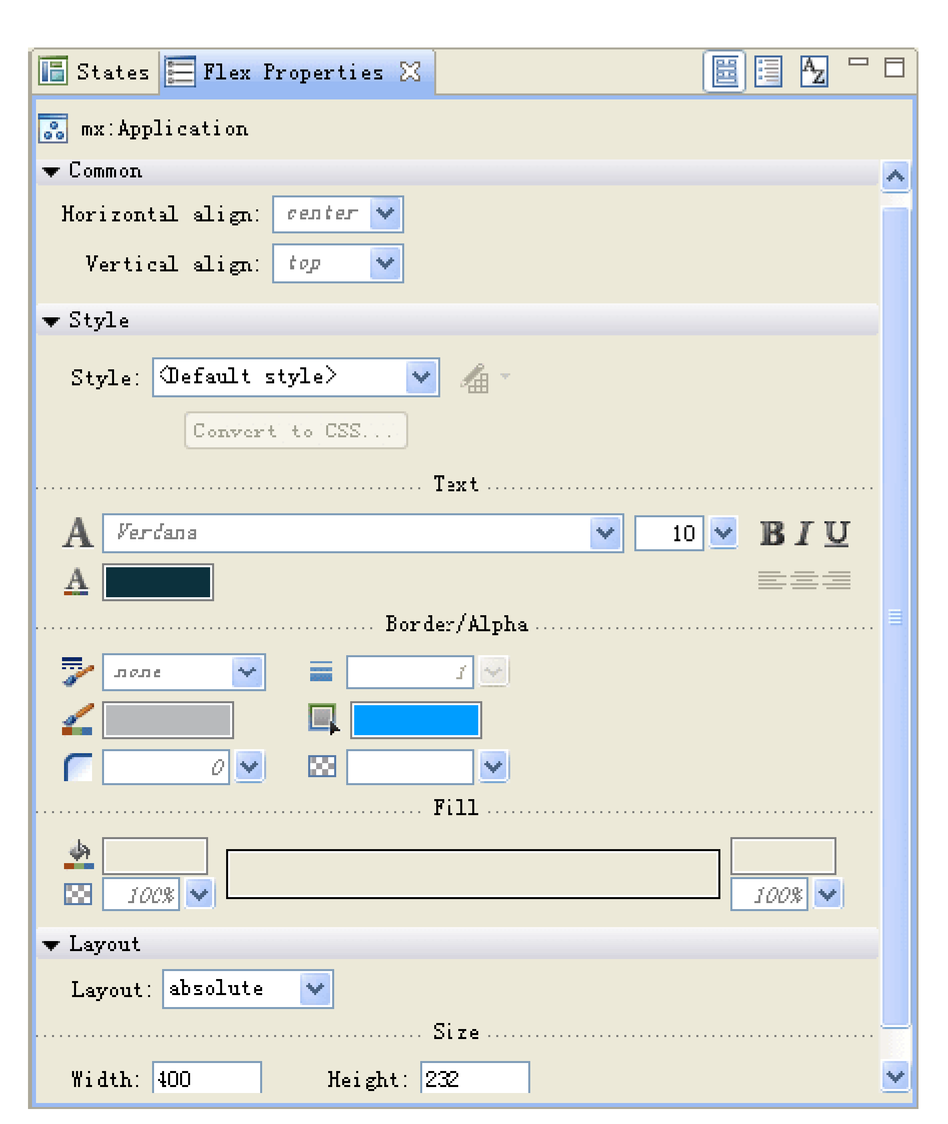
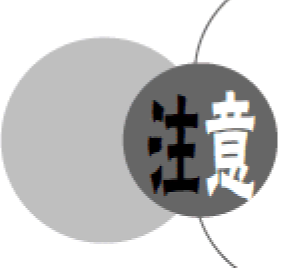


图 10-1 Application 属性面板



在属性面板的标题栏右边，有 3 个图标，分别代表了 3 种属性列表方式：标准视图、分类视图、索引视图。图 10-1 所示的是标准视图。在该视图下，只列出当前选中组件的一些常规属性，并不列出所有属性。

在属性面板中，可以设置 Application 组件的背景颜色、透明度、字体、文本颜色、文本大小等属性，为应用程序定义一个默认的显示样式。

在 Flex 中，当 Application 组件的 layout 属性为 absolute 时，对容器内的任一元素，都可以进行约束布局。约束布局（Constraint Layout）是 Flex 布局中一种非常重要的功能。切换到设计视图，在编辑区中单击任一元素，在属性面板中，都可以看到如图 10-2 所示的布局控制界面。

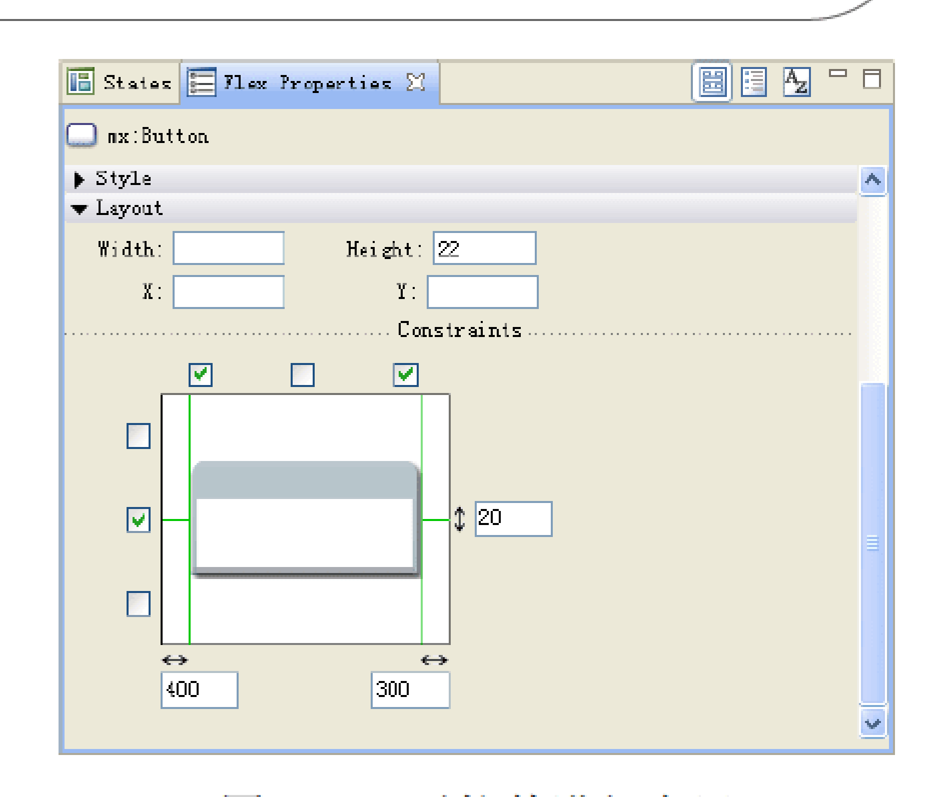


图 10-2 对控件进行布局

图 10-2 中 6 个复选框按照从左到右、从上到下的顺序分别代表左边距、水平中心距离、右边距、顶边距、垂直中心距离和底边距。各个复选

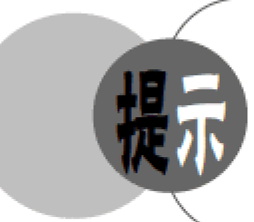
框的作用如下所示。

- **左边距**　组件左边界位置与父级容器左边界之间的水平方向距离。
- **水平中心距离**　组件的中心与父级容器中心之间的水平方向距离。
- **右边距**　组件右边界位置与父级容器右边界之间的水平方向距离。
- **顶边距**　组件上边界位置与父级容器上边界之间的垂直方向距离。
- **垂直中心距离**　组件的中心与父级容器中心之间的垂直方向距离。
- **底边距**　组件下边界位置与父级容器下边界之间的垂直方向距离。

当对某一个组件使用约束布局后，会强制定位在相应的位置，设置的 x、y 坐标将会失效。在约束定位中，中心距离和边距只能选择一个，即如果选择了水平中心距离，则水平方向的左右边距就不能同时存在，否则后面的属性将会不起作用。图 10-2 就对按钮组件使用了水平中心距离和垂直中心距离定位。该按钮的源代码如下所示：

```
<mx:Button label="提交"  click="show()" height="22" verticalCenter="20" left=
"400" right="300"/>
```

从上述代码中可以看出，left 属性代表左边距，right 属性代表右边距，verticalCenter 属性代表垂直中心距离。利用约束布局，可以实现更加特殊的网页布局功能。



约束布局只有当容器的 layout 属性值为 absolute 时才可以使用，具有这一特性的容器有：Application、Canvas、Panel 和 TitleWindow。

10.1.2　ApplicationControlBar 组件

ApplicationControlBar 组件是和 Application 相关的一个容器组件，该组件通常用来提供全局导航，如网页中的顶部导航菜单。ApplicationControlBar 组件有水平方向和垂直方向两种选择，默认为水平方向，其子级元素的布局方式也相对应与水平或者垂直。

ApplicationControlBar 组件有一个最重要的属性 dock，该属性表示是否将 Application ControlBar 组件强行停靠在应用程序界面的顶部，默认为 false。在默认情况下，Application ControlBar 组件会被当成一个普通的组件，其长度、宽度、位置都需要进行设置。如果 dock 属性为 true，ApplicationControlBar 组件将始终停靠在应用程序界面的顶部，并且其宽度始终为 100%。

下面创建了一个使用 ApplicationControlBar 组件的简单实例，具体代码 10.1 所示。

代码 10.1　使用 ApplicationControlBar 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   paddingLeft="10" paddingRight="10" paddingBottom="10"
   backgroundColor="0xCCCCCC" backgroundGradientAlphas="[0.48, 0.46]" back
   groundGradientColors="[#DCD2D2, #181717]">
   <mx:Label text="当 docked 属性值为 true 时，ApplicationControlBar 组件位于应用
```

```
        程序窗口顶部。"/>
        <mx:Spacer height="100%" />
        <mx:ApplicationControlBar width="80%">
            <mx:Label text="Normal" color="blue"/>
            <mx:Label text="查询:" />
            <mx:TextInput width="56" maxWidth="200" />
            <mx:Spacer width="100%" />
            <mx:Button label="Go itZcn.com" />
        </mx:ApplicationControlBar>
        <mx:ApplicationControlBar dock="true" paddingTop="0" paddingBottom="0">
            <mx:Label text="Docked" color="blue"/>
            <mx:MenuBar id="myMenuBar" labelField="@label" width="318">
                <mx:XMLList>
                    <menuitem label="文件" >
                        <menuitem label="打开文件" />
                        <menuitem label="保存文件" />
                    </menuitem>
                    <menuitem label="帮助"/>
                    <menuitem label="查看"/>
                    <menuitem label="操作" >
                        <menuitem label="复制" type="radio" groupName="one"/>
                        <menuitem label="粘贴" type="radio" groupName="one"/>
                        <menuitem label="剪切" type="radio" groupName="one"/>
                    </menuitem>
                </mx:XMLList>
            </mx:MenuBar>
        </mx:ApplicationControlBar>
        <mx:Label text="当 docked 属性值为 false 时，ApplicationControlBar 组件可位于应
        用程序任何位置。"/>
    </mx:Application>
```

在代码 10.1 中，插入了两个 ApplicationControlBar 组件。其中一个 ApplicationControlBar 组件设置其 dock 属性为 true，另一个设置为 false。设置其长度和宽度，并在组件内添加相应子组件。具体的演示效果如图 10-3 所示。

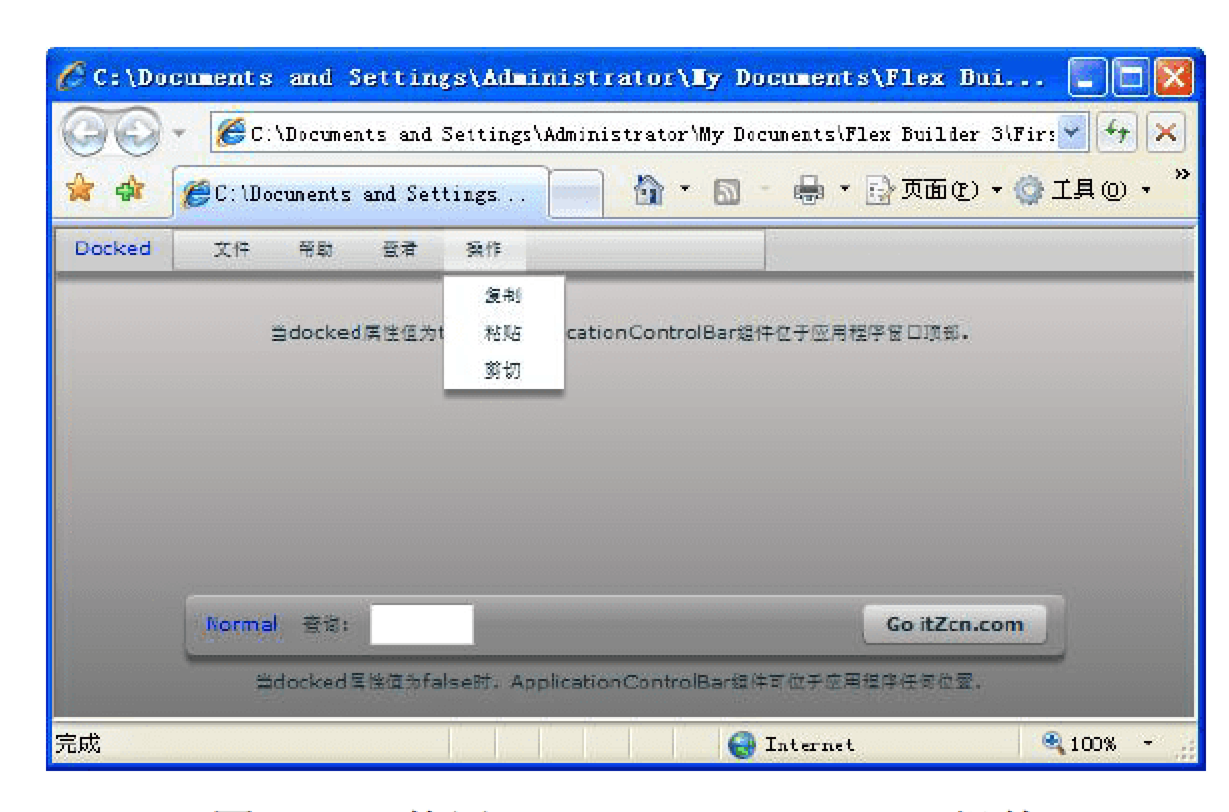


图 10-3　使用 ApplicationControlBar 组件

10.1.3　HBox、VBox 和 Canvas 组件

利用 Application 组件可以控制应用程序的总体布局，但是在实际的开发过程中，为了页面内容的显示效果考虑，通常会按照功能将界面分成若干个独立的模块，这时就需要添加其他的容器组件，并对各个模块进行相应的布局。

1. HBox 和 VBox 组件

HBox 组件和 VBox 组件都是 Box 组件的子类。Box 组件对子级元素采取规则的布局方式。其中 HBox 内的元素是水平方向分布，而 Vbox 内的元素则是垂直方向分布。

HBox 和 VBox 组件使用起来非常简单，下面就创建一个简单的使用 HBox 和 VBox 组件的实例，具体代码如代码 10.2 所示。

代码 10.2　使用 HBox 和 VBox 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" backgroundGradient
Alphas="[0.5, 0.5]" backgroundGradientColors="[#55A7C2, #3A61B0]">
<mx:Script>
    <![CDATA[
    import mx.controls.Alert;
        [Bindable]
        public var selectItem:Object;
    ]]>
</mx:Script>
   <mx:Panel title="Box Container Example" height="75%" width="75%"
      paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom="10">
      <mx:Label width="100%" color="blue"
         text="垂直排列控件" fontSize="13"/>
      <mx:VBox width="149" height="123" verticalAlign="middle" horizontal
      Align="left" borderStyle="solid">
         <mx:Button label="按钮一"/>
         <mx:Button label="按钮二"/>
         <mx:Button label="按钮三"/>
         <mx:ComboBox width="119" close="selectItem=ComboBox(event.target).
         selectedItem,Alert.show('你选择了: '+selectItem)">
           <mx:dataProvider>
           <mx:String>春天</mx:String>
           <mx:String>夏天</mx:String>
           <mx:String>秋天</mx:String>
           <mx:String>冬天</mx:String>
           </mx:dataProvider>
         </mx:ComboBox>
      </mx:VBox>
```

```
            <mx:Label width="100%" color="blue"
               text="控件水平排列" fontSize="13"/>
            <mx:HBox width="338" horizontalAlign="center" height="49" vertical
            Align="middle" borderStyle="solid">
               <mx:Button label="按钮一"/>
               <mx:Button label="按钮二"/>
               <mx:Button label="按钮三"/>
               <mx:ComboBox width="115" close="selectItem=ComboBox(event.target).
               selectedItem,Alert.show('你选择了: '+selectItem)">
               <mx:dataProvider>
                 <mx:String>春天</mx:String>
                 <mx:String>夏天</mx:String>
                 <mx:String>秋天</mx:String>
                 <mx:String>冬天</mx:String>
                 </mx:dataProvider>
               </mx:ComboBox>
            </mx:HBox>
        </mx:Panel>
    </mx:Application>
```

在代码 10.2 中，分别设置了 VBox 和 HBox 组件的 verticalAlign、horizontalAlign 属性，并且设置两个组件的边框样式。在浏览器中运行，具体的效果如图 10-4 所示。

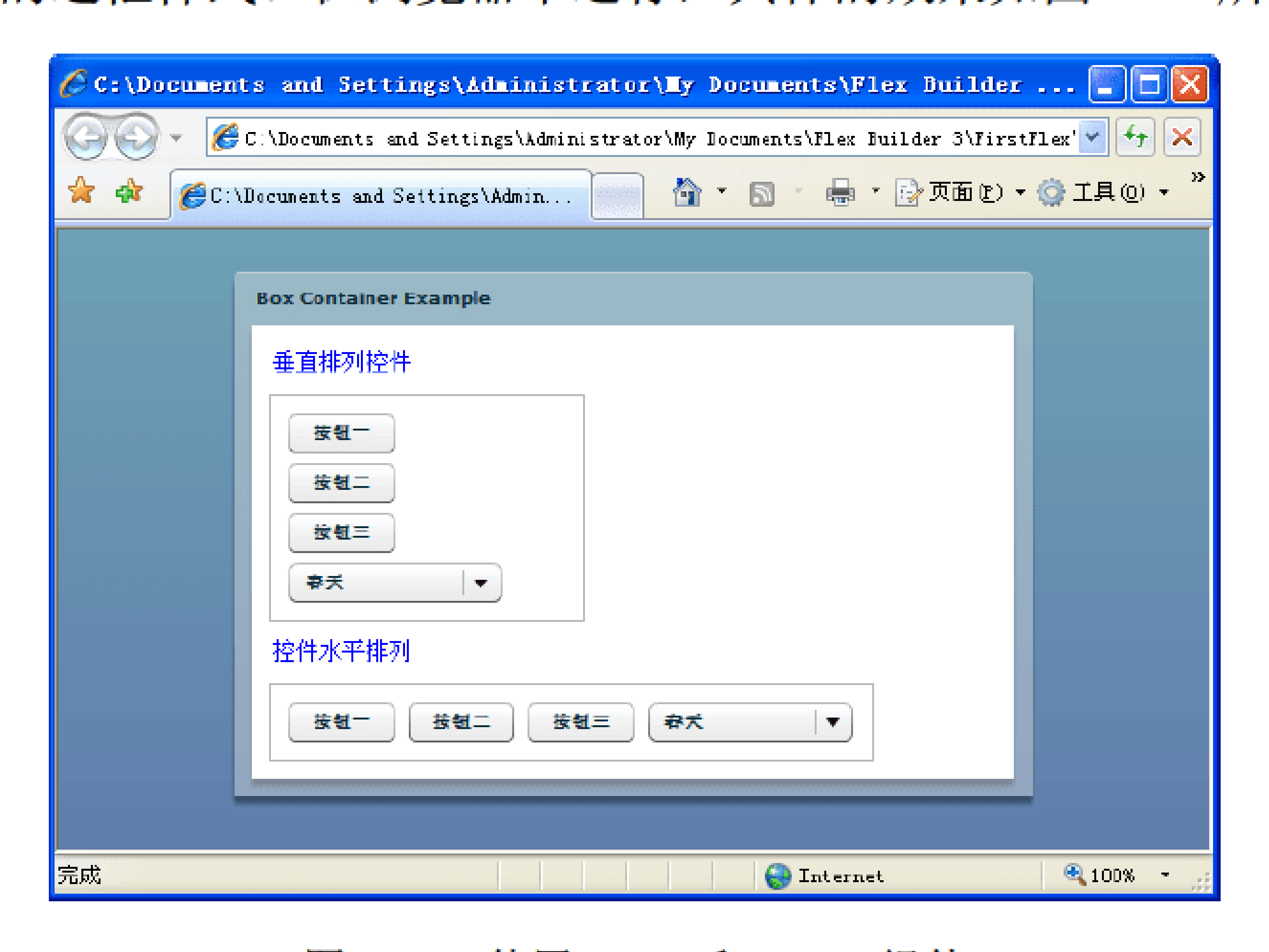


图 10-4　使用 HBox 和 VBox 组件

2．Canvas 组件

Canvas 组件直接继承自 Container 类，和其他容器比较起来，具有体积小、使用灵活等特点。与前面介绍的 HBox 组件和 VBox 组件不同，放置在 Canvas 容器组件里的元素只能由 x 和 y 的坐标来定位，也就是说 Canvas 的 layout 属性固定为 absolute。如果不指定坐标，元素的 x、y 坐标默认为 0。

通过设置 Canvas 组件的其他属性，还可以改变具体的显示效果。例如，修改容器内文本

颜色、字体样式、背景颜色等。

下面创建一个使用 Canvas 容器的实例，具体如代码 10.3 所示。

代码 10.3　使用 Canvas 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Panel title="Canvas Container Example" height="95%" width="95%"
      paddingTop="5" paddingLeft="5" paddingRight="5" paddingBottom="5"
      layout="absolute">
      <mx:Label width="100%" color="blue"
         text="使用绝对定位布局容器" fontSize="12" x="5" y="5"/>
      <mx:Canvas borderStyle="solid" height="200" width="100%" x="5" y="31">
         <mx:Text width="100" x="12" y="5" text="体积小"/>
         <mx:Text width="100" x="91" y="30" text="使用灵活"/>
         <mx:Text width="100" x="158" y="55" text="绝对定位"/>
         <mx:VBox width="75" height="78" backgroundColor="#8080C0" alpha=
         "0.8" x="159" y="113"/>
         <mx:VBox width="75" height="78" backgroundColor="#FFFF80" x="10"
         y="64">
         </mx:VBox>
         <mx:VBox width="75" height="78" backgroundColor="#0080C0" x="85" y=
         "90">
         </mx:VBox>
      </mx:Canvas>
      </mx:Panel>
</mx:Application>
```

在代码 10.3 中，在 Canvas 容器中拖曳进了几个简单的控件，并在 Canvas 中定义子级元素的文本显示颜色和大小，以及容器的背景颜色等。具体的效果如图 10-5 所示。

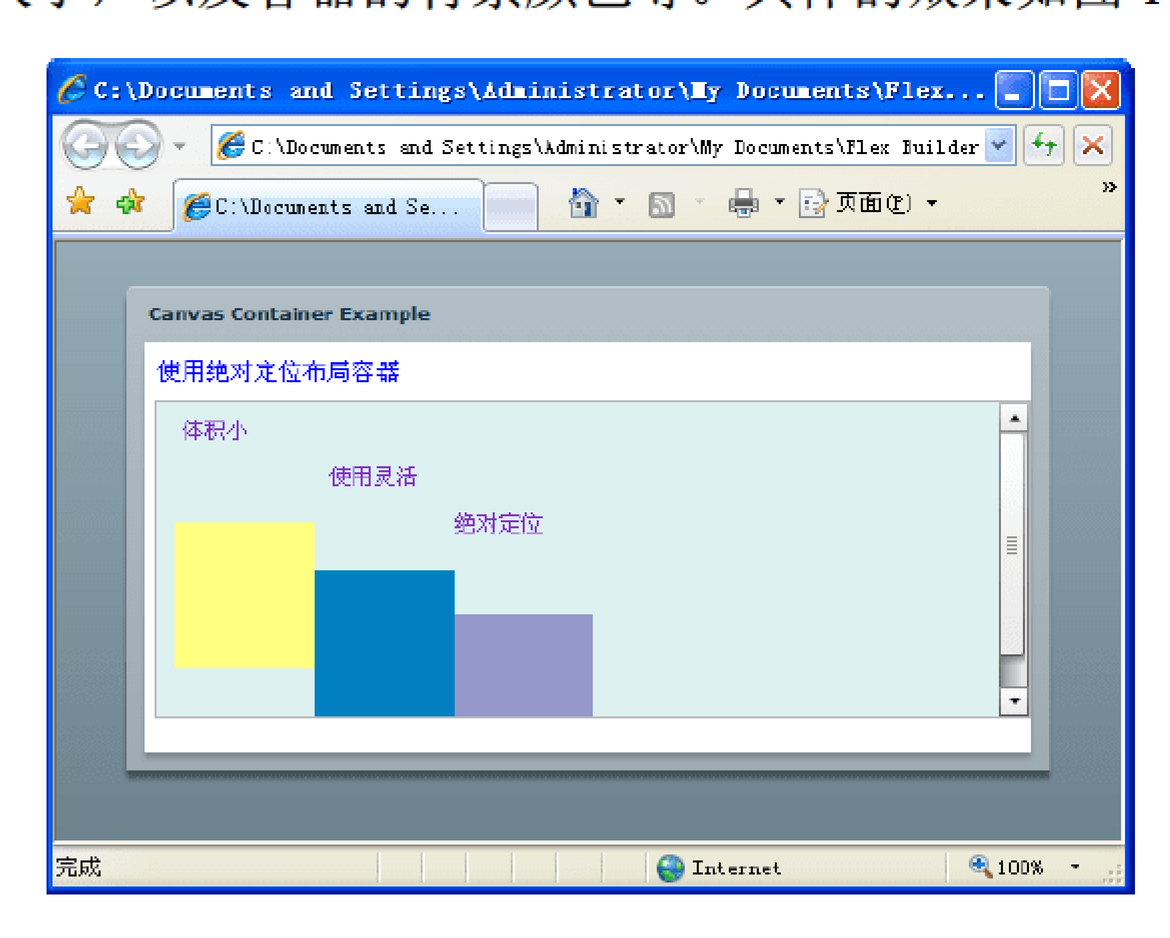


图 10-5　使用 Canvas 组件

当 Canvas 容器中组件位置超出 Canvas 的区域范围时，Canvas 组件自动增加滚动条。每

个容器都自带了滚动条，默认状态下滚动条采用自动适应的原则，只有容器的内容超出容器的可视范围才会显示。当然也可以通过修改 horizontalScrollPolicy 属性和 verticalScrollPolicy 属性来控制滚动条的显示方式。这两个属性有 3 个可选值：on 表示滚动条总是出现，off 表示滚动条总是不出现，auto 表示只有在需要的时候才出现，是默认状态。在开发过程中，可以根据具体的需要，控制容器滚动条的显示方式。

10.1.4 HDividedBox 和 VDividedBox 组件

HDividedBox 和 VDividedBox 组件继承自父类 DividedBox。与 HBox 和 VBox 组件相似，这两个组件也是对包含的子级元素采用规则的布局方式，只不过多了一个功能，那就是在子级元素之间增加了可以拖动的分割块。在运行程序时，通过拖动分割块可以动态调整分割块附近元素的长度或者宽度。

HDividedBox 和 VDividedBox 组件分别对页面进行水平切割和垂直切割，通过 verticalGap 和 horizontalGap 属性可以分别调整子级容器之间的间隔，即分割块间的宽度，默认情况下为 10 像素。

HDividedBox 和 VDividedBox 组件的边框属性 borderStyle 默认值为 none，表示不显示边框。当然用户可以根据实际需求，设置显示边框。例如设置该属性值为 solid，表示采用实心线条作为边框。

HDividedBox 和 VDividedBox 组件还有一个比较常用的属性 liveDragging，该属性是一个 Boolean 类型。当为 false 时，表示在拖动分割块时候，分割块附近的元素只在鼠标松开时才调整位置；如果为 true，则在拖动时就会不断的调整位置。默认为 false。

下面创建一个使用 HDividedBox 和 VDividedBox 组件分割页面的实例，该实例实现一个简易电子相册的功能，如代码 10.4 所示。

代码 10.4　制作电子相册

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
initialize="flash.system.System.useCodePage=true" creationComplete="Operate
XML()" horizontalAlign="center" verticalAlign="middle">
    <mx:Script>
      <![CDATA[
       import mx.messaging.AbstractConsumer;
         [Bindable]
         public var selectedNode:XML;
         [Bindable]
         public var treenew:XML;
         // Event handler for the Tree control change event.
         public function treeChanged(event:Event):void {
            if(Tree(event.target).selectedItem.@data!=""){
            selectedNode=Tree(event.target).selectedItem as XML;
            }
```

```
        }
      public function OperateXML():void{
       var testloader:URLLoader=new URLLoader();
       var req:URLRequest=new URLRequest("tree.xml")
       testloader.dataFormat=URLLoaderDataFormat.TEXT;
       testloader.addEventListener(Event.COMPLETE,handleComplete);
       testloader.load(req);
      }
      public function handleComplete(event:Event):void
      {
      try
      {
       treenew=new XML(event.target.data);
      }
      catch(e:TypeError)
      {
       trace(e.message);
      }
}
 ]]>
  </mx:Script>
   <mx:HDividedBox width="99%" height="98%">
       <mx:Panel height="99%" layout="absolute" width="200" title="相片列
       表" fontSize="12">
           <mx:Tree  id="tree1" width="100%" dataProvider="{treenew}" label
           Field="@label"height="100%"change="treeChanged(event)"showRoot
           ="false">
           </mx:Tree>
       </mx:Panel>
       <mx:VDividedBox height="99%">
           <mx:Panel height="80%" layout="vertical" width="100%" horizontal
           Align="center" verticalAlign="middle">
        <mx:VBox horizontalGap="0" verticalGap="0">
           <mx:HBox horizontalGap="0">
               <mx:Image source="images/1_01.gif" />
               <mx:Image source="images/1_02.gif" />
               <mx:Image source="images/1_03.gif" />
           </mx:HBox>
           <mx:HBox horizontalGap="0">
               <mx:Image source="images/1 04.gif" />
               <mx:Image horizontalAlign="center" width="500" height="300"
               source="{selectedNode.@data}" includeInLayout="true"/>
               <mx:Image source="images/1_05.gif" />
           </mx:HBox>
           <mx:HBox horizontalGap="0">
```

```
                <mx:Image source="images/1_06.gif" />
                <mx:Image source="images/1_07.gif" />
                <mx:Image source="images/1_08.gif" />
            </mx:HBox>
        </mx:VBox>
        </mx:Panel>
        <mx:TabNavigator width="100%" height="20%" backgroundColor="#B8E6EE"
        backgroundAlpha="0.4" borderStyle="solid" alpha="1.0" borderColor=
        "#8F8B8B">
         <mx:Canvas label="正在播放" width="100%" height="100%">
         <mx:TextArea width="100%" height="100%" text="{selectedNode.
         @label}">
               </mx:TextArea>
         </mx:Canvas>
        </mx:TabNavigator>
       </mx:VDividedBox>
   </mx:HDividedBox>
</mx:Application>
```

在代码 10.4 中，使用 HDividedBox 和 VDividedBox 组件对应用程序界面进行布局，并在相应组件中嵌套使用了其他容器组件。在应用程序的左侧，使用了 Tree 组件，引用外部 XML 文件作为数据源，显示电子相册列表，右侧使用了 VBox 组件和 HBox 组件嵌套，并使用 Image 组件制作了一个相框。当用户选择列表中的某一个图片名称时，在相框中显示相应的图片信息并在 TextArea 组件中显示正在播放的图片名称。具体的显示效果如图 10-6 所示。

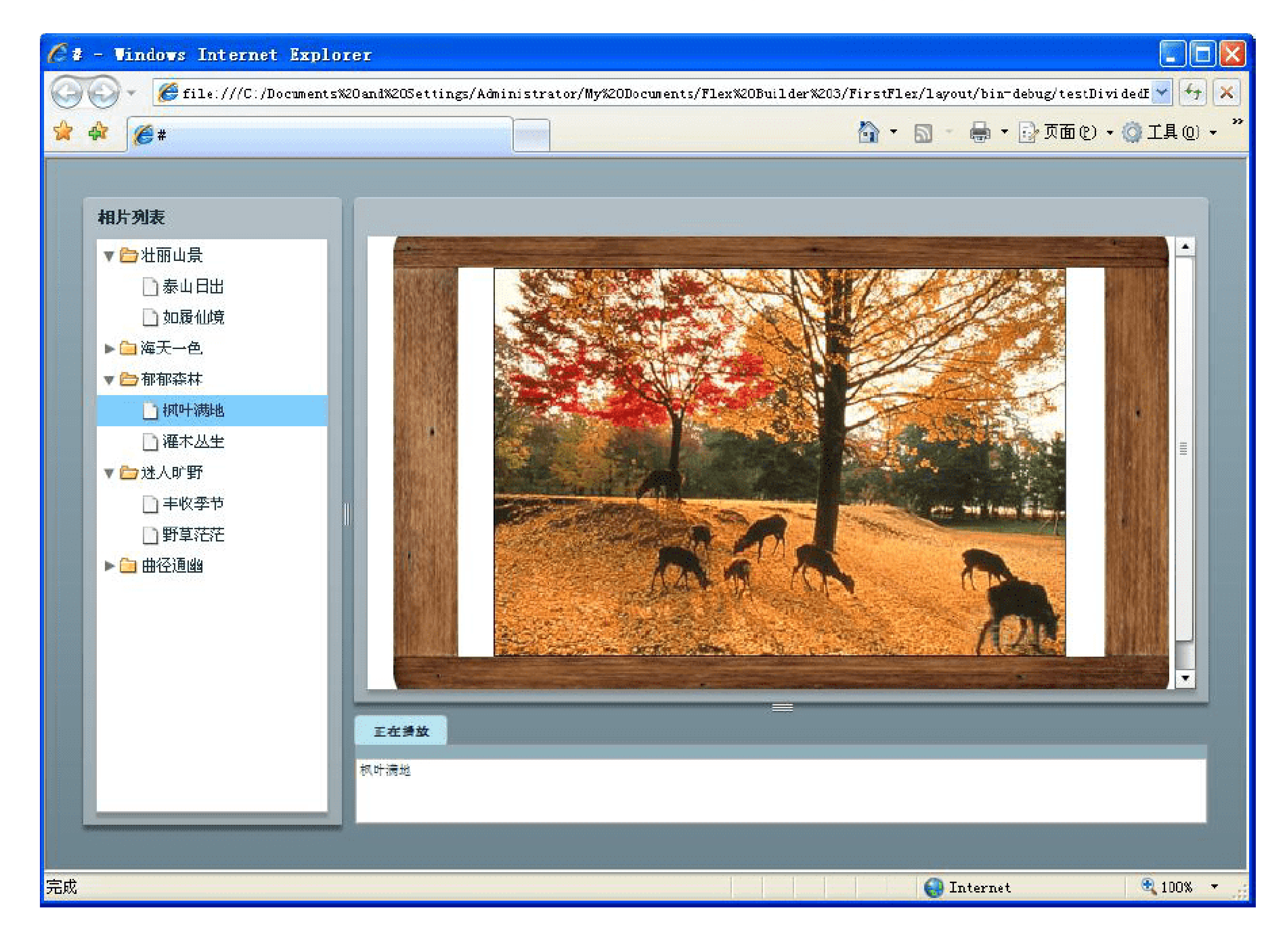


图 10-6　电子相册

10.2　窗口布局

在应用程序中，窗口是最常见的表现方式，使用窗口布局可以使界面更加丰富、更加直观，用户操作起来也会非常方面。本节将会详细介绍两种与窗口布局有关的组件：Panel 组件和 TitleWindow 组件。

10.2.1　Panel 组件

Panel 组件是应用程序开发过程中最常用的组件之一，在前面的实例中已经多次用到了 Panel 组件。该组件具有 Canvas 和 HBox、VBox 组件的所有功能。如果 Panel 组件的 layout 属性值为 absolute，则 Panel 组件对子级元素的布局方式和 Canvas 组件一样；当值为 horizontal 时与 HBox 组件布局方式一样；当值为 vertical 时则与 VBox 组件布局方式完全相似。

使用 Panel 组件非常简单，只需向容器内拖曳进相应的子级组件即可。表 10-1 列出了 Panel 组件的一些常用属性。

表 10-1　Panel 组件的常用属性

属性	说明
id	唯一标识 Panel 组件
title	窗口的标题
width	窗口的宽度
height	窗口的高度
layout	Panel 组件的布局方式
visible	该 Panel 组件是否可见
backgroundColor	定义容器的背景颜色
backgroundImage	定义容器的背景图像
color	定义容器内的文本颜色
fontSize	定义容器内的文本大小
fontFamily	定义容器内的文本字体

下面使用 Panel 组件设计了一个简单的用户登录界面，如代码 10.5 所示。

代码 10.5　设计用户登录窗口

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
backgroundGradientAlphas="[0.2, 0.5]" backgroundGradientColors="[#51A378,
#5CB1D4]">
    <mx:Script>
        <![CDATA[
            import mx.controls.Alert;
            private function show():void
```

```
                {
                    if(username.text=="" || pwd.text==""){
                      Alert.show("用户名和密码不能为空");
                    }
                    else
                    {
                      if(username.text=="admin" && pwd.text=="admin")
                      {
                      Alert.show("登录成功");
                      }
                      else
                      Alert.show("用户名称或者密码错误，请确认");
                    }
                }
            ]]>
        </mx:Script>
        <mx:Panel width="316" height="189" layout="absolute" title="用户登录"
        fontSize="12" horizontalCenter="0" verticalCenter="8">
            <mx:Label text="用户名: " fontSize="10" y="30" x="43"/>
            <mx:Label text="密  码: " fontSize="10" y="68" x="43"/>
            <mx:TextInput id="username" fontSize="10" y="28" x="86"/>
            <mx:TextInput id="pwd" fontSize="10" y="66" x="86" displayAsPassword=
            "true"/>
            <mx:Button label="确定" fontSize="10" y="101" x="68" click="show()"/>
            <mx:Button label="取消" fontSize="10" y="101" x="161" />
        </mx:Panel>
    </mx:Application>
```

在代码 10.5 中，使用了一个 Panel 组件，设置其布局属性 layout 的属性值为 absolute、标题为“用户登录”、文本字体大小为 12，并使用约束布局进行定位。在容器内，使用 Label 组件、TextInput 组件和 Button 组件创建了一个简单的用户登录界面，并对用户输入的结果进行处理。具体的显示效果如图 10-7 所示。

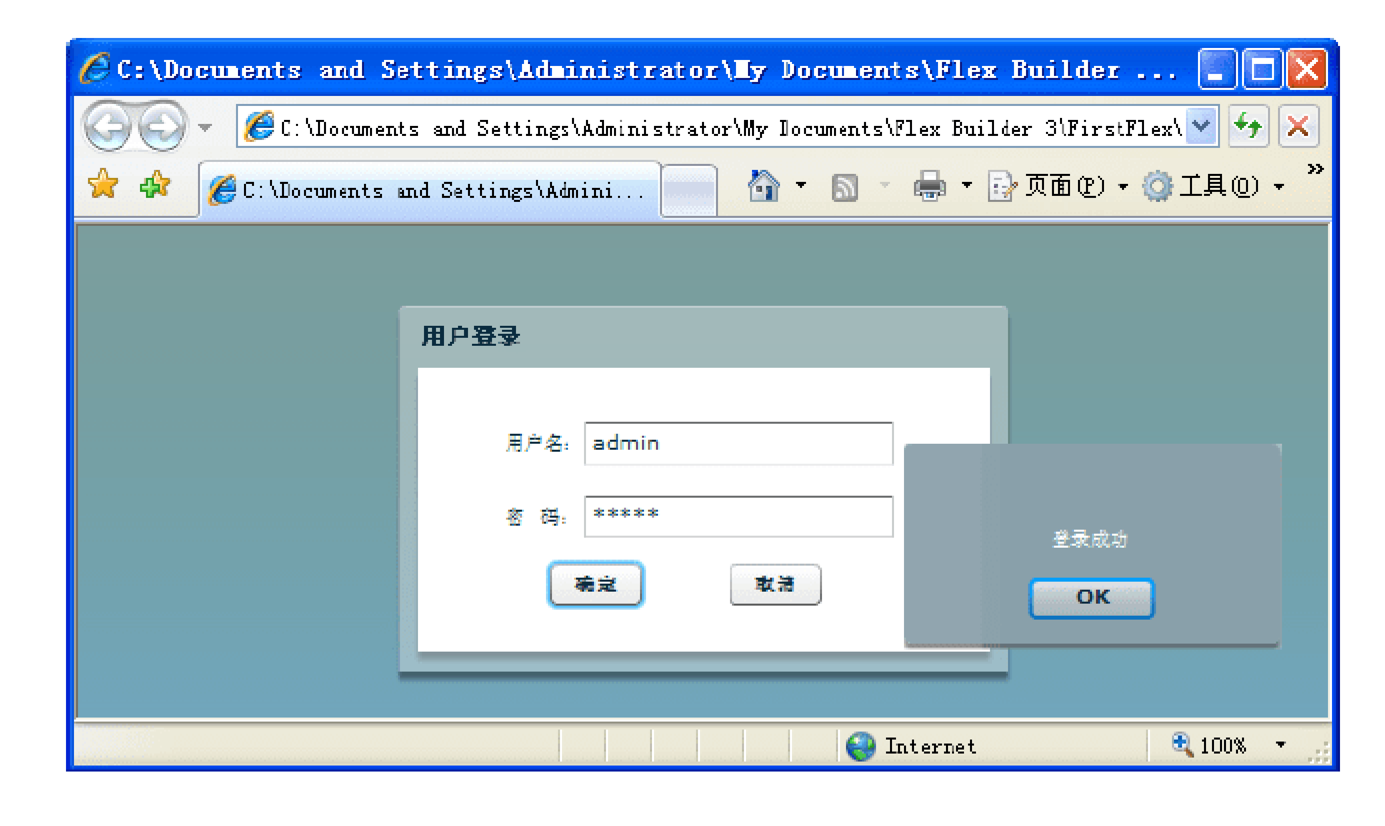


图 10-7　用户登录界面

和 Application 一样，Panel 也有一个相关的容器 ControlBar。该组件与 ApplicationControl Bar 组件功能相似。不同的是，ControlBar 组件位置不可以调整，总是被固定在 Panel 组件的底部，并且与 Panel 组件等宽。与 ApplicationControlBar 组件相似，ControlBar 组件中对子级组件采用水平排列的布局方式。

例如，给上面实例中的 Panel 组件添加 ControlBar 组件，添加一个注册按钮，如代码 10.6 所示。

代码 10.6 添加 ControlBar 组件

```
<mx:Panel width="316" height="205" layout="absolute" title="用户登录" fontSiz
e="12" horizontalAlign="center" horizontalCenter="0" verticalCenter="8">
    <mx:Label text="用户名: " fontSize="10" y="30" x="43"/>
    <mx:Label text="密  码: " fontSize="10" y="68" x="43"/>
    <mx:TextInput id="username" fontSize="10" y="28" x="86"/>
    <mx:TextInput id="pwd" fontSize="10" y="66" x="86" displayAsPassword
    ="true"/>
    <mx:Button label="确定" fontSize="10" y="101" x="68" click="show()"/>
    <mx:Button label="取消" fontSize="10" y="101" x="161" />
    <mx:ControlBar horizontalAlign="right" height="25" y="162" fontSize=
    "10" width="3">
        <mx:LinkButton label="单击这里注册" color="#EA1446"/>
    </mx:ControlBar>
</mx:Panel>
```

在代码 10.6 中，在 Panel 组件内添加了一个 ControlBar 组件，并且在 ControlBar 组件内添加 LinkButton 组件，设置 ControlBar 组件的对齐方式为右对齐。具体的显示效果如图 10-8 所示。

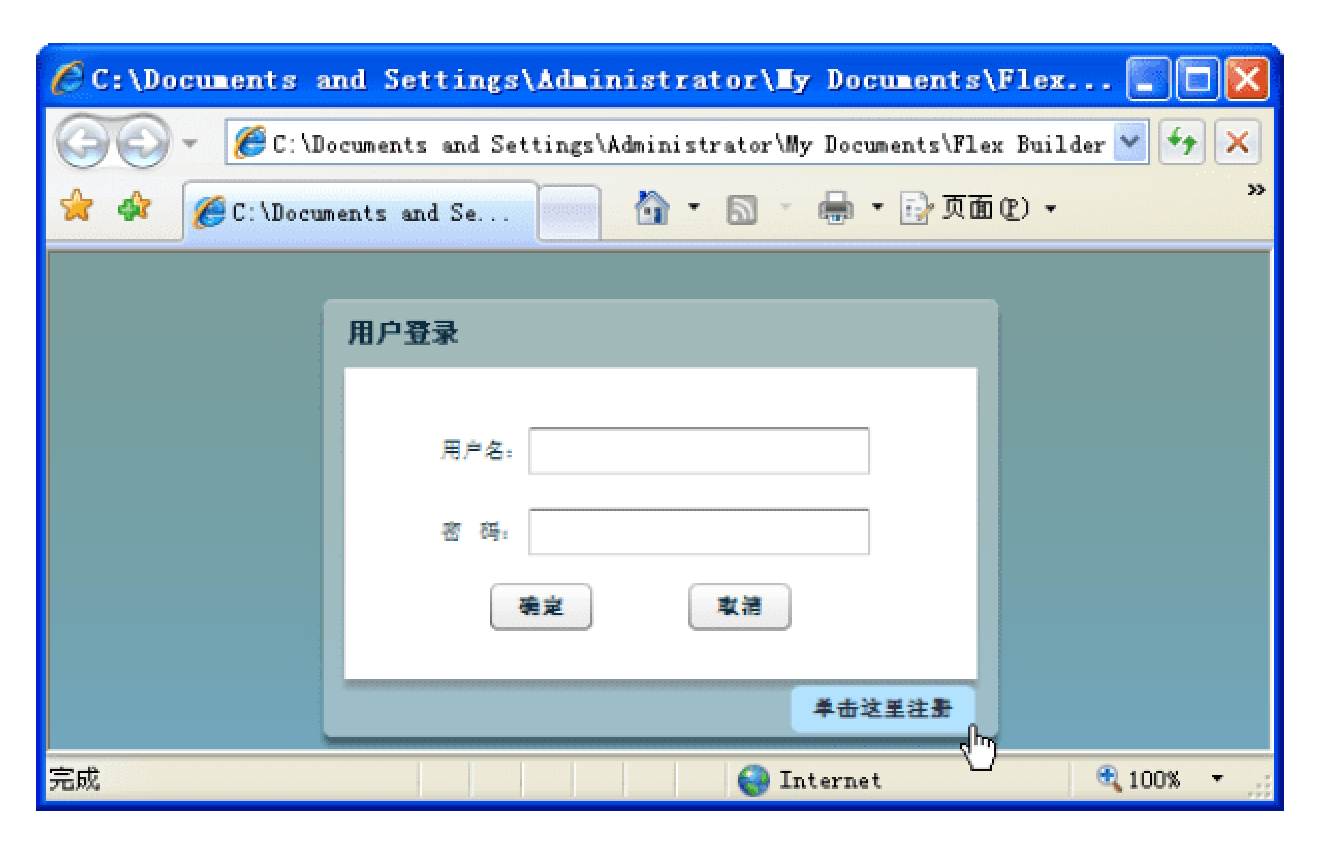
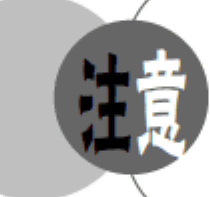


图 10-8 添加注册按钮



在代码 10.6 中，设置 ControlBar 组件的 width 属性值为 3，但是 ControlBar 组件宽度与 Panel 组件等宽，将忽视该属性。

10.2.2 TitleWindow 组件

TitleWindow 组件继承于 Panel 组件，与 Panel 组件相比，只是多了一个【关闭】按钮。默认情况下，TitleWindow 组件不带【关闭】按钮，在开发过程中需要的话，可以设置其 showCloseButton 属性为 true，显示【关闭】按钮。在运行应用程序时，用户单击【关闭】按钮时，将触发 TitleWindow 组件的 close 事件，可以对该事件进行监听和处理。

下面创建一个使用 TitleWindow 组件的实例，首先创建 MXML Application 程序，如代码 10.7 所示。

代码 10.7　创建 Application 程序

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;
            import mx.containers.TitleWindow;
            import flash.geom.Point;
            private var point1:Point = new Point();
            private function showWindow():void {
             var  login:SimpleTitleWindowExample=SimpleTitleWindowExample(Pop
             UpManager.createPopUp( this, SimpleTitleWindowExample , true));
                point1.x=myButton.x;
                point1.y=myButton.y;
                point1=myButton.localToGlobal(point1);
                login.x=point1.x+25;
                login.y=point1.y+25;
                login.loginName=returnedName;
            }
        ]]>
    </mx:Script>
    <mx:Panel title="TitleWindow Container Example" height="75%" width="75%"
    paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom="10" font
    Size="10">
        <mx:Button id="myButton" label="单击按钮打开 TitleWindow 容器" click=
        "showWindow();" fontSize="10"/>
        <mx:Text id="returnedName" text="" width="100%"/>
    </mx:Panel>
</mx:Application>
```

在代码 10.7 中，使用 Panel 组件进行布局，再使用一个 Button 按钮和 Text 文本框。当用户单击按钮时，调用函数 showWindow()。在函数 showWindow()中，使用 PopUpManager 组件的 CreatePopUp()方法打开 TitleWindow 组件。PopUpManager 是专门处理弹出窗口的对象。

PopUpManager.createPopUp()方法创建弹出窗口，其 3 个参数分别代表：对弹出窗口所基于的窗口的引用、对要创建的对象的类的引用和弹出窗口是否为模态窗口（位于程序最顶层）。

接下来，编写主程序 SimpleTitleWindowExample.mxml，具体代码如代码 10.8 所示。

代码 10.8　创建 TitleWindow 元件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" title="Title
Window" x="168" y="86" showCloseButton="true" fontSize="13" close="PopUp
Manager.removePopUp(this);">
    <mx:Script>
        <![CDATA[
            import mx.managers.PopUpManager;
            import mx.controls.Text;
            public var loginName:Text;
            private function returnName():void {
                loginName.text="你的名字为：" + userName.text;
                PopUpManager.removePopUp(this);
            }
        ]]>
    </mx:Script>
    <mx:HBox fontSize="10">
        <mx:Label text="输入你的名称："/>
        <mx:TextInput id="userName" width="100%"/>
    </mx:HBox>
    <mx:HBox fontSize="10">
        <mx:Button label="确定" click="returnName();"/>
        <mx:Button label="取消" click="PopUpManager.removePopUp(this);"/>
    </mx:HBox>
</mx:TitleWindow>
```

在代码 10.8 中，在 TitleWindow 容器内使用 Label、TextInput 和 Button 组件，提示用户输入姓名，并在函数 retrunName()中对用户的输入做出相应处理。在程序中，【取消】按钮的 click 事件和 TitleWindow 组件的 close 事件都调用 PopUpManager 对象的 removePopUp()方法，关闭 TitleWindow 界面。具体的显示效果如图 10-9 所示。

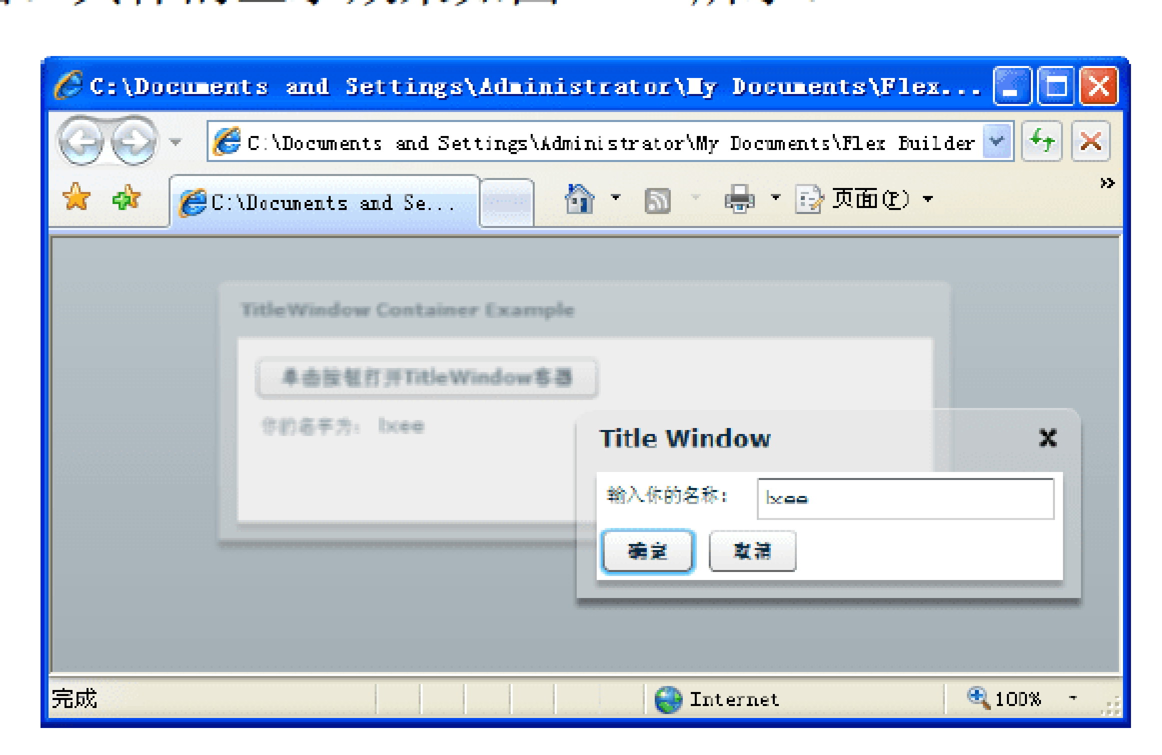


图 10-9　使用 TitleWindow 组件

10.3 表单布局

表单是网页中最常用的元素之一，利用表单可以接收用户输入的数据并进行相应的处理。例如完成用户注册、编写日志、调查问卷等功能。在 Flex Builder 3 中，提供了一套表单组件，包括 Form 组件、FormItem 组件和 FormHeading 组件。

Form 组件是表单功能组件中的主要部分，继承自 Container 对象，在 Form 组件内可以拖曳进其他控制组件。FormHeading 组件主要用于显示表单标题，当然也可以放置顶部导航控制组件等。

Form 容器是由若干个 FormItem 组件组成的。当用户向 Form 表单容器中拖曳进来一个控制组件时，默认产生一个 FormItem 组件对象。该组件有两个重要属性：label 和 required。label 属性显示提示文本，required 属性指定该栏的值是否可以为空，这在处理用户输入时非常有用。

下面使用 Form 组件创建一个统计用户信息的表单，如代码 10.9 所示。

代码 10.9　创建表单

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Panel title="Form Container Example" paddingTop="10" paddingLeft="10"
    paddingRight="10" paddingBottom="10" layout="absolute" width="440"
    verticalCenter="13" horizontalCenter="-1" height="332">
        <mx:Form width="382" height="228" horizontalCenter="0" vertical
        Center="-19">
            <mx:FormHeading label="用户资料"/>
            <mx:FormItem label="用户名: " required="true">
                <mx:TextInput id="fname" width="200"/>
            </mx:FormItem>
            <mx:FormItem label="生日 (mm/dd/yyyy): " required="true">
                <mx:TextInput id="dob" width="200"/>
            </mx:FormItem>
            <mx:FormItem label="性别: " width="333">
            <mx:HBox>
              <mx:RadioButton label="男" groupName="1" selected="true"/>
                <mx:RadioButton label="女" groupName="1"/>
            </mx:HBox>
            </mx:FormItem>
            <mx:FormItem label="电子邮件: " required="true">
                <mx:TextInput id="email" width="200"/>
            </mx:FormItem>
            <mx:FormItem label="地址">
                <mx:TextInput id="adress" width="200"/>
            </mx:FormItem>
```

```
                <mx:FormItem label="联系方式" required="true">
                    <mx:TextInput id="phone" width="200"/>
                </mx:FormItem>
            </mx:Form>
            <mx:Button label="提交" verticalCenter="125" horizontalCenter="-49"/>
            <mx:Button label="取消" verticalCenter="125" horizontalCenter="45"/>
        </mx:Panel>
        <mx:StringValidator source="{fname}" property="text" minLength="4"
        maxLength="12"/>
        <mx:PhoneNumberValidator source="{phone}" property="text"/>
        <mx:DateValidator source="{dob}" property="text"/>
        <mx:EmailValidator source="{email}" property="text"/>
    </mx:Application>
```

在代码 10.9 中，在 Panel 容器中创建了一个统计用户信息的表单。设置相应 FormItem 组件的 required 属性为 true。在运行程序时，就会在相应 FormItem 的显示文本后出现红色的*符号，提示用户该行信息不可以为空。这里设置用户名、生日、电子邮件和联系方式信息不可以为空。

在程序尾部，添加了 4 个验证对象。其中 StringValidator 用于字符验证，PhoneNumber Validator 用于电话号码验证，DateValidator 用于日期验证，EmailValidator 用于邮箱地址验证。这几个验证组件有一个 source 属性，表示需要验证的目标对象。在后面的章节中会详细讲解有关数据验证的知识，这里就不多介绍。

运行程序，具体效果如图 10-10 所示。

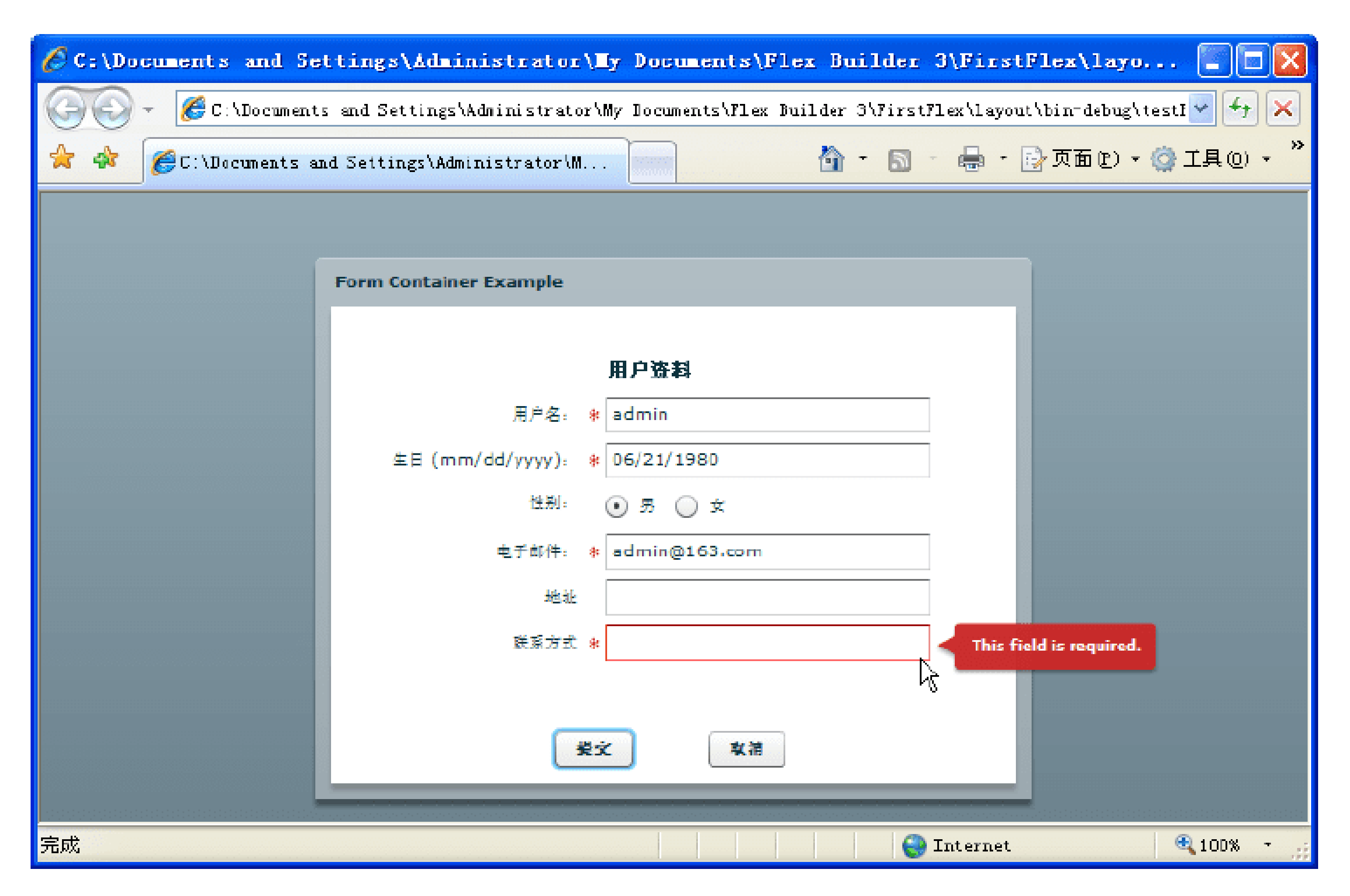


图 10-10　使用表单

10.4　动态控制对象的布局

在前面介绍过的容器中，其子级元素的定位都是相对静止。如果需要添加一个新的组件，

则必须指定该组件的起始 x、y 轴坐标值，这对于大型的应用程序界面来说，将非常困难。本节就介绍两种可以实现动态控制对象显示位置的组件：Tile 组件和 Grid 组件。

10.4.1 Tile 组件

Tile 组件直接继承自 Container 对象，在 Tile 容器中，对子级元素按照水平或者垂直方向动态进行排列，并且每一个子级元素都处于一个相同大小的单元格中，Tile 组件就通过控制单元格的个数进行规则布局。

Tile 组件有几个重要的属性：direction、tileWidth 和 tileHeight。其中 direction 属性有两个值，horizontal 和 vertical，分别代表设置其子级元素的布局方向为水平方向还是垂直方向。tileWidth 和 tileHeight 属性用来控制每个单元格的长度和宽度。

下面创建一个使用 Tile 组件排列多个子级组件的实例，并设置当用户单击按钮后，动态添加一个新的按钮，如代码 10.10 所示。

代码 10.10 使用 Tile 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
    <![CDATA[
        import mx.controls.Button;
        private var i:int=7;
        private function addBut():void{
       var btn:Button=new Button();
        btn.label=i.toString();
        btn.height=40;
        btn.width=75;
        t1.addChild(btn);
        i++;
        }
    ]]>
</mx:Script>
   <mx:Panel title="Tile Container Example"
      paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom="10">
      <mx:Label width="100%" color="blue"
         text="Tile 容器将子级元素布局在平等大小的的单元格中。" fontSize="11"/>
      <mx:Tile id="t1" direction="horizontal" borderStyle="inset" horizontal
      Gap="10" verticalGap="15" paddingTop="10" paddingBottom="10" padding
      Left="10" paddingRight="10" width="294" height="182">
         <mx:Button label="1" height="40" width="75"/>
         <mx:Button label="2" height="40" width="75"/>
         <mx:Button label="3" height="40" width="75"/>
         <mx:Button label="4" height="40" width="75"/>
         <mx:Button label="5" height="40" width="75"/>
```

```
                <mx:Button label="6" height="40" width="75"/>
            </mx:Tile>
            <mx:ControlBar horizontalAlign="right" height="34" verticalAlign=
            "top">
                <mx:LinkButton label="增加新按钮" fontSize="12" click="addBut()"/>
            </mx:ControlBar>
        </mx:Panel>
    </mx:Application>
```

在代码 10.10 中，设置 horizontalGap 为 10、verticalGap 为 15，即两个单元格之间的水平间隔和垂直间隔分别为 10 和 15 像素。这里并没有设置每个单元格的长度和宽度，那么就默认按照最大的组件的长度和宽度进行计算。当用户单击【增加新按钮】命令时，在最后一个单元格的后面自动添加一个新的按钮。具体的效果如图 10-11 所示。



图 10-11　使用 Tile 组件布局

10.4.2　Grid 组件

在使用 Tile 组件时，很难精确控制每一行的对象个数，因为 Tile 组件需要对组件的宽度和子级元素的宽度进行计算，还要考虑元素的间隔问题，十分麻烦。在 Flex Bulider 3 中，就可以使用另外一个动态布局组件：Grid 组件。

Grid 组件继承自 Box 对象，类似于网页中的表格，由若干行组成，每行包含若干个单元格，每个单元格中可以包含其他元素。在 Grid 组件中，GridRow 对象表示行，GridItem 对象表示单元格。GridRow 对象和 GridItem 对象都继承自 HBox 对象。

在 Grid 组件中，每个单元格的高度和宽度可以各不相同，但是相同的行的单元格的高度必须相同。如果一行中每个单元格中的元素高度不同，则以高度值最大的单元格的高度为准。同样，相同的列的宽度值也必须相同。如果一列中每个单元格的宽度不相同，则以宽度值最大的单元格的宽度为准。

与表格工具一样，每个单元格具有两个重要的属性：colSpan 和 rowSpan，分别表示单元

格所占的横向格数和纵向格数。

下面创建了一个使用 Grid 容器组件的简单实例，在该实例中，创建了一个 3 行 3 列的表格，每个单元格中拖曳进去一个 Button 按钮组件，并设置一些单元格的 colSpan 和 rowSpan 属性，如代码 10.11 所示。

代码 10.11　使用 Grid 组件

```
<?xml version="1.0"?>
<mx:Application  borderStyle="none"  xmlns:mx="http://www.adobe.com/2006/
mxml">
    <mx:Panel title="Grid Container Example" paddingTop="10" paddingLeft="10"
    paddingRight="10" paddingBottom="10">
        <mx:Label width="100%" color="blue" text="一个三行三列的网格容器" fontSize
        ="12"/>
        <mx:Grid borderStyle="solid">
            <mx:GridRow>
                <mx:GridItem borderStyle="solid">
                    <mx:Button label="第一行第一列" width="110" height="30"/>
                </mx:GridItem>
                <mx:GridItem  horizontalAlign="left" verticalAlign="top"
                colSpan="2" rowSpan="1" borderStyle="solid">
                    <mx:Button label="第一行第二列" width="100"/>
                </mx:GridItem>
                <mx:GridItem borderStyle="solid">
                    <mx:Button label="第一行第三列" width="100"/>
                </mx:GridItem>
            </mx:GridRow>
            <mx:GridRow>
                <mx:GridItem borderStyle="solid">
                    <mx:Button label="第二行第一列" width="100"/>
                </mx:GridItem >
                <mx:GridItem rowSpan="2" borderStyle="solid">
                    <mx:Button label="第二行第二列" width="100"/>
                </mx:GridItem>
                <mx:GridItem borderStyle="solid" horizontalAlign="left"
                verticalAlign="top" colSpan="1" rowSpan="1">
                    <mx:Button label="第二行第三列" width="100"/>
                </mx:GridItem>
            </mx:GridRow>
            <mx:GridRow>
                <mx:GridItem borderStyle="solid">
                    <mx:Button label="第三行第一列" width="100"/>
                </mx:GridItem>
                <mx:GridItem borderStyle="solid">
                    <mx:Button label="第三行第二列" width="100"/>
```

```
                </mx:GridItem>
                <mx:GridItem borderStyle="solid">
                    <mx:Button label="第三行第三列" width="100"/>
                </mx:GridItem>
            </mx:GridRow>
        </mx:Grid>
    </mx:Panel>
</mx:Application>
```

运行上面的程序，其具体效果如图 10-12 所示。从图中可以看出，这里一共有 3 行 3 列，每个单元格中的元素并不完全相同，但是每一行和每一列的高度和宽度都相同，并且自动对齐。由于第 1 行第 2 列单元格中的 colSpan 属性值为 2，即横向占了两个单元格的宽度，第 1 行第 3 个单元格将自动后移；第 2 行第 2 列单元格的 rowSpan 属性为 2，即纵向占用两个单元格的高度，那么第 3 行的第 2 个和第 3 个单元格将自动后移。

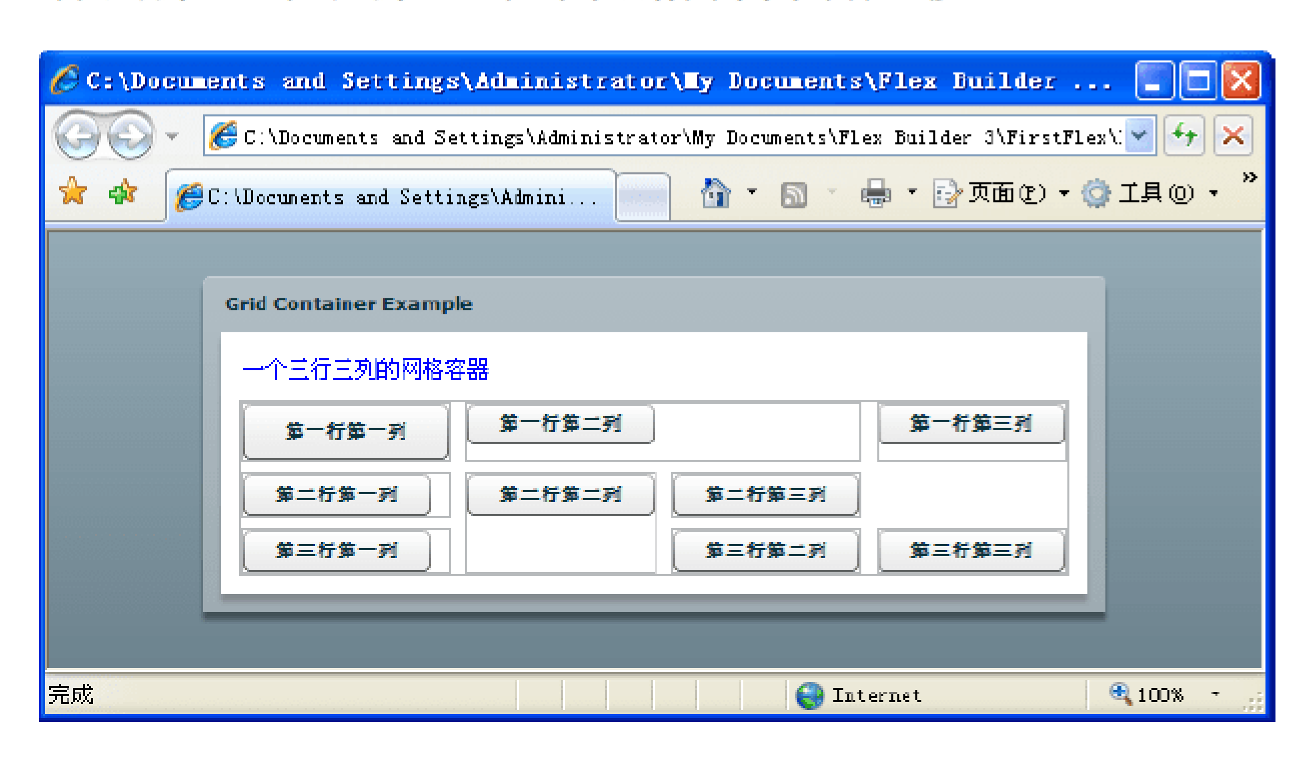


图 10-12　创建 3 行 3 列的网格

在 Grid 容器组件中，默认行与行之间、列与列之间存在着间隔。如果需要取消间隔，则可以设置 Grid 组件的 horizontalGap 和 verticalGap 属性值为 0。例如，修改代码 10.11 中的间隔值为 0，其显示效果如图 10-13 所示。

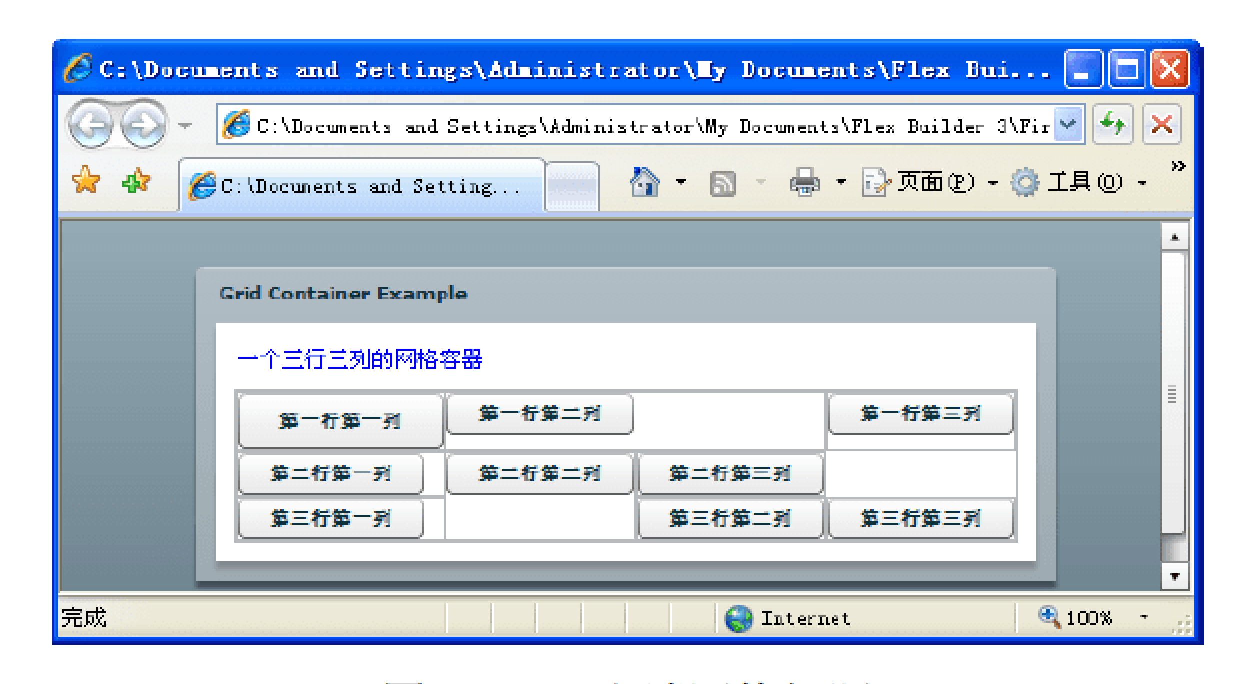


图 10-13　取消网格间隔



在 Grid 组件，默认水平间隔为 8 像素，默认垂直间隔为 6 像素。

10.5 导航容器

在浏览网站时，经常可以看到许多内容模块都使用了导航容器，可以在一个局部模块间切换内容，即丰富了页面的显示内容，又为页面添加了视觉上的动态效果。在 Flex Builder 3 中也提供了这样的导航容器。本节将详细介绍这类组件。

10.5.1 ViewStack 组件

ViewStack 组件是由若干个重叠在一起的子容器组成的，每次只有一个容器是可见的或者活动的。ViewStack 组件的子容器可以包括 Canvas、Panel、HBox、VBox 等。

ViewStack 容器本身并没有为用户提供在不同的子容器视图间切换的功能，但是可以通过 ActionScript 脚本语言来进行控制，或者把 ViewStack 组件与其他控制类组件结合在一起使用，如 ButtonBar、LinkBar 组件等。

ViewStack 组件的 SelectedChild 属性表示当前处于激活状态的子级容器对象。通过控制该属性的值，就可以实现在不同的子容器间切换的功能。

下面创建了一个使用 ViewStack 组件的简单实例，在该实例中，使用 Button 按钮控制显示 ViewStack 组件的子级容器，如代码 10.12 所示。

代码 10.12　使用 ViewStack 组件

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Panel title="ViewStack Container Example" height="95%" width="95%"
       paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom
       ="10">
      <mx:Text width="100%" color="blue" text="单击按钮进入相应的版块"/>
      <mx:HBox borderStyle="solid" width="100%" paddingTop="5" paddingLeft
      ="5" paddingRight="5" paddingBottom="5">
          <mx:Button id="but_sports" label="体育版块"
              click="myViewStack.selectedChild=sports;"/>
          <mx:Button id="but_music" label="音乐版块"
              click="myViewStack.selectedChild=music;"/>
          <mx:Button id="but_news" label="新闻版块"
              click="myViewStack.selectedChild=news;"/>
      </mx:HBox>
      <mx:ViewStack id="myViewStack" borderStyle="solid" width="100%" height
      ="80%">
          <mx:Canvas id="sports" backgroundColor="#FFFFCC" label="sports"
          width="100%" height="100%">
              <mx:Label text="这里是体育版块" color="#000000"/>
          </mx:Canvas>
```

```
            <mx:Canvas id="music" backgroundColor="#CCFFFF" label="music" width
            ="100%" height="100%">
                <mx:Label text="这里是音乐版块" color="#000000"/>
            </mx:Canvas>
            <mx:Canvas id="news" backgroundColor="#FFCCFF" label="news" width=
            "100%" height="100%">
                <mx:Label text="这里是新闻版块" color="#000000"/>
            </mx:Canvas>
        </mx:ViewStack>
    </mx:Panel>
</mx:Application>
```

在代码 10.12 中，创建了一个 ViewStack 组件，并在里面添加了 3 个 Canvas 容器组件，并设置每个 Canvas 容器的内容及背景颜色。在 ViewStack 组件上面，创建了 3 个 Button 按钮，分别设置其 click 事件，在事件中设置 ViewStack 组件的 selectedChild 属性值，用来显示相应的子容器。具体效果如图 10-14 所示。

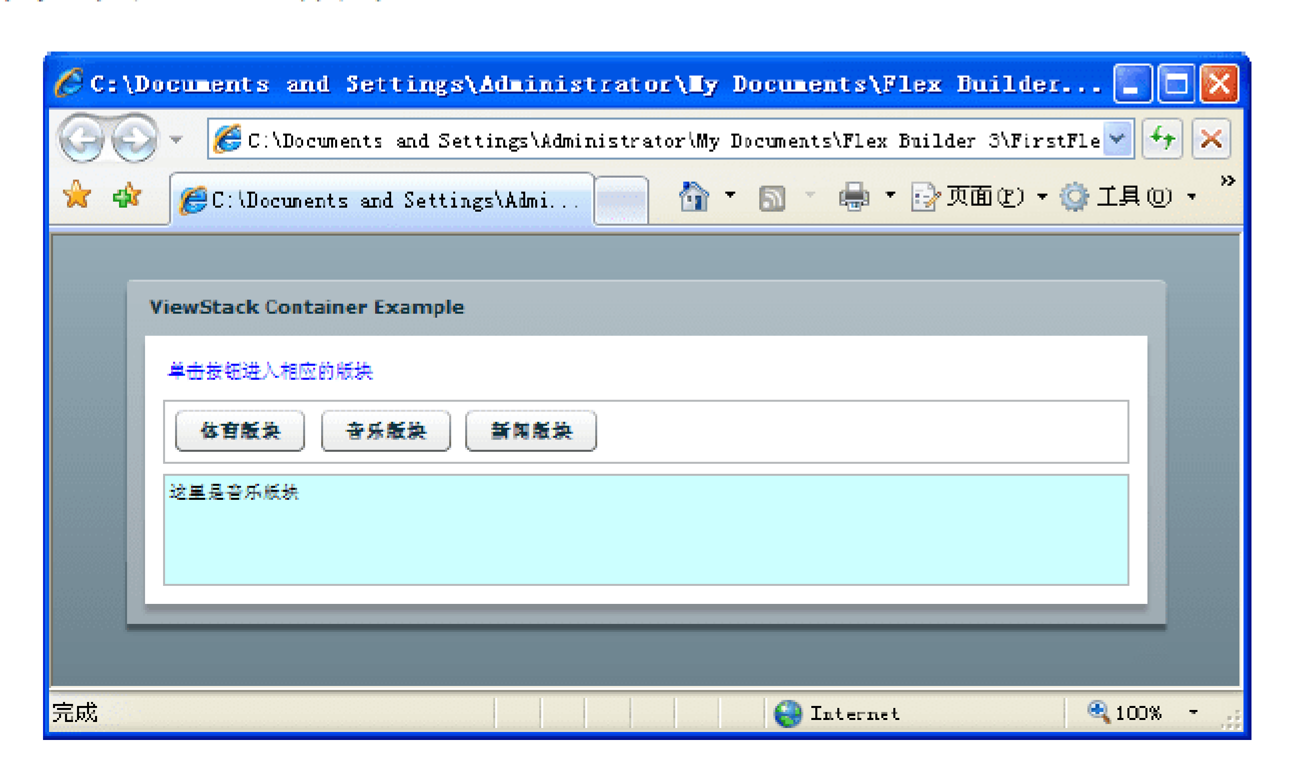


图 10-14　使用 ViewStack 组件

10.5.2　Accordion 组件

Accordion 组件是一个可折叠的导航容器，与 ViewStack 组件不同，该组件包含一个子面板的列表，但是每次只显示一个面板。如果要切换到相应的子面板，单击与之相对应的导航按钮即可。在实际的网站开发过程中，Accordion 组件应用非常广泛，用户可以按任何顺序访问子面板，随意在子面板中前后移动。

在 Accordion 组件的子级容器中，需要设置其 label 属性，该属性的值即为 Accordion 组件的导航按钮的显示文本。

下面创建了一个使用 Accordion 组件的实例，该实例显示新闻图片展示功能，如代码 10.13 所示。

代码 10.13　新闻图片展示

```
<?xml version="1.0"?>
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Panel title="图片新闻" height="90%" width="90%"
        paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom="10">
        <mx:HBox width="598">
          <mx:VBox>
            <mx:Accordion id="accordion" width="260" height="370" verticalGap
           ="0">
              <mx:VBox label="游泳健将" width="100%" verticalAlign="top">
                <mx:Image source="images/111.jpg" width="250" height="175"/>
                <mx:TextArea text="看看这体格，多么健壮，特别是两个黑色泳镜，简直帅呆
                了！" width="250"/>
              </mx:VBox>
              <mx:VBox label="小小教书匠" width="100%">
                <mx:Image source="images/113.jpg" width="250" height="194"/>
                <mx:TextArea text="看这位老师是第一次走上讲台吧，看这下面才几个学生，
              脸就红成这样了！" width="100%"/>
              </mx:VBox>
              <mx:VBox label="铜牙利齿" width="100%">
                <mx:Image source="images/114.jpg" width="250" height="176"/>
                <mx:TextArea text="好家伙，真够厉害的啊，连钢制的刀具都想要咬断啊？"
                width="100%"/>
              </mx:VBox>
              <mx:VBox label="大力士" width="100%" verticalGap="0">
                <mx:Image source="images/115.jpg" width="250" height="171"/>
                <mx:TextArea text="看看这细胳膊细腿的，竟然能搬这个大堆的货物，真是一
              个大力士" width="100%"/>
              </mx:VBox>
              <mx:VBox label="苹果蝴蝶" width="100%">
                <mx:Image source="images/116.jpg" width="250" height="197"/>
                <mx:TextArea text="见过苹果蝴蝶没有？看看它正在吃苹果呢。" width=
                "100%"/>
              </mx:VBox>
            </mx:Accordion>
          </mx:VBox>
          <mx:VBox width="330" height="100%">
              <mx:TextArea width="100%" height="50%" text="       幽默一刻
  一个韩国人、一个日本人和一个中国人看见，巴西队得冠军后回国时飞机有战斗机护航，都很羡
  慕。他们就一起去问上帝。韩国人问：韩国队什么时候，可以得世界冠军？上帝想了一会说：你
  这辈子可以看到。日本人就问：日本队什么时候可以得世界冠军？上帝想了一会说：你这辈子是
  看不到了，我还是可以看到的。最后中国人问：中国队什么时候可以得世界冠军？上帝想了半天，
  突然大哭到：我这辈子也看不到了….. " fontSize="12">
            </mx:TextArea>
            <mx:TextArea width="100%" height="50%" text="搞网络的     蜘蛛和蜜蜂
            结婚了。蜘蛛感到很难过，就问它的妈妈：你为何一定要我娶蜜蜂？蜘蛛的妈妈说：蜜蜂
            是吵了点，但人家好歹也是个空姐。蜜蜂也感到很难过，也问它的妈妈：为什么要让我嫁
```

```
                给蜘蛛呢？蜜蜂的妈妈说：蜘蛛是丑了一点，但人家好歹也是搞网络的..."fontSize=
                "12">
                </mx:TextArea>
              </mx:VBox>
            </mx:HBox>
        </mx:Panel>
    </mx:Application>
```

在代码 10.13 中，创建了一个 Accordion 组件，在组件内创建了 5 个 VBox 容器组件，在每个子容器中添加 Image 组件和 TextInput 组件。设置每个 VBox 组件的 label 属性、Image 组件的 source 属性和 TextInput 组件的 text 属性，并且设置每个组件的宽度和高度。运行程序，单击 Accordion 组件中的导航按钮，就可以在不同的新闻图片之间进行切换。具体的效果如图 10-15 所示。



图 10-15　新闻图片展示

10.5.3　TabNavigator 组件

TabNavigator 组件继承自 ViewStack 组件，因此它拥有 ViewStack 组件的所有特性。并且在 ViewStack 组件的基础上，添加了可以供用户切换内容的界面接口。在 TabNavigator 组件的顶部增加了一个标签条，单击标签条上面的标签，就可以切换到相对应的容器面板。

和 Accordion 组件相似，TabNavigator 组件中每一个子容器都需要设定其 label 属性，该属性的值即为标签提示文本。

下面使用 TabNavigator 组件创建了一个简单的实例，模拟创建网页中一个具体的计算机教程模块，如代码 10.14 所示。

代码 10.14　创建教程内容模块

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
backgroundGradientAlphas="[0.5, 0.9]" backgroundGradientColors="[#79CDDA,
#155174]">
  <mx:XMLList id="flex">
    <node label="第 1 章 认识 Flex3.0">
        <node label="关于 ActionScript 3.0" />
        <node label="开发环境——Flash CS3 环境配置" />
        <node label="开发环境——Flex Builder 环境搭建" />
    </node>
    <node label="第 2 章  熟悉开发环境 Flex Builder 3">
        <node label="熟悉 Flex Builder 3 工作区" />
        <node label="编译与运行 Flex 3.0 程序详解" />
        <node label="调试 Flex 3.0 程序" />
        <node label="Flex 3.0 项目概述" />
        <node label="Flex Builder 3 常用快捷键" />
    </node>
    <node label="第 3 章  ActionScript 3.0 语法">
        <node label="变量和常量"/>
        <node label="数据类型" />
        <node label="运算符" />
        <node label="流程控制语句"/>
    </node>
    <node label="第 4 章  函数与数组">
        <node label="函数"/>
        <node label="数组" />
    </node>
  </mx:XMLList>
  <mx:Panel width="406" height="271" layout="absolute" title="计算机视频教程"
  verticalCenter="-16" horizontalCenter="5">
    <mx:TabNavigator width="100%" height="100%">
        <mx:Canvas width="100%" height="100%" label="Asp 教程">
        </mx:Canvas>
        <mx:Canvas label="Jsp 教程" width="100%" height="100%">
        </mx:Canvas>
        <mx:Canvas label="Flex 教程" width="100%" height="100%">
        <mx:Tree id="t1" dataProvider="{flex}" labelField="@label" width=
        "100%" height="100%"/>
        </mx:Canvas>
        <mx:Canvas label="Asp.net 教程" width="100%" height="100%">
        </mx:Canvas>
        <mx:Canvas label="Java 教程" width="100%" height="100%">
        </mx:Canvas>
    </mx:TabNavigator>
  </mx:Panel>
</mx:Application>
```

在代码 10.14 中，使用 TabNavigator 组件，在组件中使用了 5 个 Canvas 容器组件，分别设置其 label、width 和 height 属性的值，并且设计每个子容器的内容。在实际应用中需要对每个子容器中的内容进行详细设计，这里由于篇幅问题，只设计了一个子容器的内容。在 label 属性为“Flex 教程”的 Canvas 容器组件中，使用了一个 Tree 组件，设置其数据源为上面定义的 XML 文件，并且设置 labelField="@label"，即解析 XML 文件中节点的 label 属性的值。

运行上面的程序，单击【Flex 教程】标签，切换到相应的容器面板，具体显示效果如图 10-16 所示。

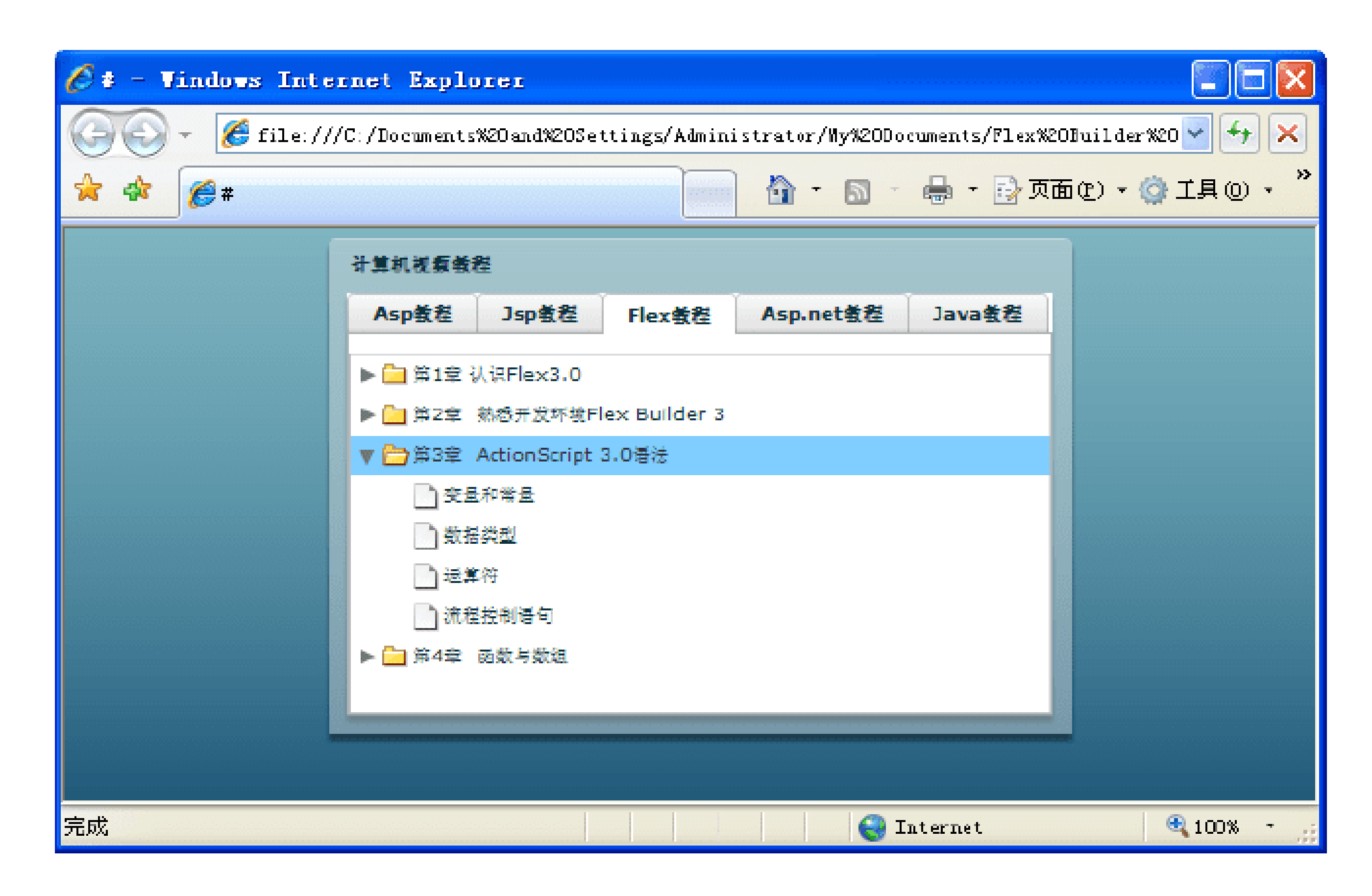
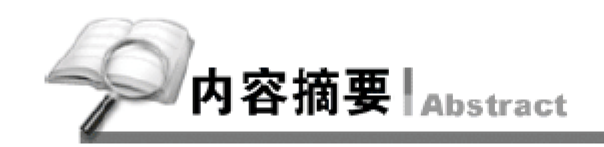


图 10-16　计算机教程模块

第 11 章 使用行为对象和动画效果



在前面章节中已经介绍了大量的 Flex 组件，本章将主要介绍 Flex 中行为和特效类组件，包括 Effects、States 和 Transition 等。通过对这些组件的学习，并结合前面的基础组件可以制作出各种动画效果，使页面的表现形式更加丰富。



- 理解行为对象的概念
- 掌握如何创建行为
- 掌握 Flex 的组件
- 熟练掌握常见动画效果的制作方法
- 掌握 State 对象的使用方法
- 掌握 Transtion 对象的使用方法

11.1 认识行为对象

首先来探讨一下什么是行为对象，理解行为对象的概念对以后的学习有很大的帮助。接下来对如何创建行为对象进行简单介绍。

11.1.1 行为对象简介

一个行为对象，可以看作是触发器（Trigger）和动画效果（Effect）的结合体。一个动画就是一个行为对象，即一个动画实际上分为触发器和动画效果这两部分。触发器是一个用户动作，往往伴随一个事件，如鼠标的单击、鼠标的松开。一旦动作发生，触发器被激活，动画就开始播放。动画效果由非可视化组件完成。Flex 中包括多种动画效果，如淡入淡出效果、放大、缩小、移动、遮罩效果等，这些动画效果都是由 ActionScript 实现的，只运行一段时间。在 Flex 中这些都被封装起来，包括常用的动画效果也被封装起来。所以使用起来非常简单。但是，触发器和事件并不相同，针对一个事件的触发器，受到事件的制约。事件可以被监听，而触发器不可以。

综上所述，行为对象允许对一个触发器定义多个动画效果，当触发器被激活，这一系列

的动画会按某种顺序运行。例如，在单击一个按钮后，图片的尺寸变大，同时图片的位置也不断变化。

行为对象适用于一切可视化的组件，而且组件已经预留了相关的实现接口，可以很方便地将一些使用频繁的事件和行为对象联系起来使用，极大地增强程序的交互性。

11.1.2　创建行为对象

行为对象的创建可以通过 MXML 和 ActionScript 两种方式来定义。下面分别对这两种方式做简单介绍。

第一种方式使用 Move 组件，在该组件中将 target 指定的作用目标 img 图片从 xFrom 移动到 xTo 位置，duration 表示持续的时间（单位为毫秒）。实例如代码 11.1 所示。

代码 11.1　使用 MXML 标签创建行为：EffectsControl.mxml

```
<mx:Move id="MoveImg" target="{img_Nokia}" xFrom="100" xTo="500" duration=
"3000"/>
<mx:Image id="img_Nokia" x="200" y="200" source="@Embed(source='../images/
Nokia_6630.png')"/>
<mx:Button label="移动位置" click="MoveImg.play()" x="200" y="150"/>
```

所有动画对象都继承 Effect 对象。抽象类定义的控制动画的播放流程的方法如下所示。

- **play(targets:Array = null, playReversedFromEnd:Boolean=false)**　开始播放。其中 targets 表示一组目标对象，这将覆盖原来的目标对象。playReversedFromEnd 指示是否逆向播放。
- **end()**　停止播放动画。
- **pause()**　暂停播放动画。
- **resume()**　当暂停时，继续播放。
- **reverse()**　逆向播放动画，如果动画正在播放，则从当前位置开始逆向播放。

也可以使用 ActionScript 来完成行为对象的创建，具体示例如代码 11.2 所示。

代码 11.2　使用 ActionScript 创建行为

```
<mx:Script>
    <![CDATA[
        import mx.effects.Move;

        private var MoveImg:Move;
        internal function init():void{
            MoveImg = new Move();
            MoveImg.xFrom =100;
            MoveImg.xTo = 500;
            MoveImg.duration = 3000;
            MoveImg.target = img_Nokia
        }
    ]]>
```

```
</mx:Script>
<mx:Image id="img_Nokia" x="200" y="200" source="@Embed(source='../images/
Nokia_6630.png')"/>
<mx:Button label="移动位置" click="MoveImg.play()" x="200" y="150"/>
```

使用以上两种方式创建的行为对象动画效果完全一致。

11.2 行为和组件

本节首先简单介绍组件的行为触发器，然后通过一个实例来详细介绍如何为组件添加行为。

11.2.1 组件的行为和动画效果

UIComponent 是所有组件的父类，该类定义了组件共有的属性和方法，其中包括行为触发器。组件常见的行为触发器如表 11-1 所示。

表 11-1 组件常见的行为触发器

触发器名称	对应的事件名称	事件描述
addedEffect	Added	当组件被添加到容器中时触发
createCompleteEffect	createComplete	当组件完成绘制时触发
removedEffect	removed	当组件从容器中删除时触发
focusInEffect	focusIn	当组件获得光标焦点时触发
focusOutEffect	focusOut	当组件失去光标焦点时触发
hideEffect	hide	当组件变成不可见时触发
showEffect	show	当组件变成可见时触发
mouseDownEffect	mouseDown	当鼠标在组件上按下时触发
mouseUpEffect	mouseUp	当鼠标在组件上松开时触发
rollOutEffect	rollOut	当鼠标从组件上移开时触发
rollOverEffect	rollOver	当鼠标移到组件上时触发
moveEffect	move	当组件被移动时触发
resizeEffect	resize	当组件大小改变时触发

在 Flex 中，所有的动画效果都是 Effect 类的子类，位于 mx.Effects 包中，这里面包含的 Effect 组件如表 11-2 所示。这些动画效果一起构成了 Flex 丰富的效果库。

表 11-2 Effect 组件

组件名称	动画效果描述
AnimateProperty	针对组件的一个以数字计算的属性，例如长度，按给定的起始值逐渐改变属性的大小
Blur	模糊效果，可以让组件变得模糊不清。核心由 BlurFilter 滤镜完成。当对组件使用了该效果，不可再使用 Blur 滤镜和其他模糊效果

续表

组件名称	动画效果描述
Fade	淡入淡出效果。注意：当目标对象中包括文字时，必须使用嵌入字体
Dissolve	溶解效果，主要是在目标对象上增加覆盖层，改变覆盖层的透明度，达到让目标消失或出现的效果。和 Fade 效果相比，它可以设置覆盖层的颜色。注意：当目标对象中包括文字时，必须使用嵌入字体
Glow	发光效果，使用了 GlowFilter 滤镜。当对组件使用了该效果，不可再使用 GlowFilter 滤镜和其他发光效果
Iris	彩虹效果，组件以矩形方式，从中心放大，或缩小到中心，属于遮罩效果
Move	移动效果。移动组件的坐标，只有当组件位于支持绝对定位的容器中时才有效，如 Canvas、Application、Panel 等
Pause	停止，什么都不改变，没有动画。一般用于复合动画效果中，用来分割前后两个动画
Resize	尺寸调整效果，改变组件的长和宽。当改变组件的长和宽时，处于同一个容器的其他组件的大小也可能会相应改变，如果该容器使用了绝对定位则不会发生这种情况
Rotate	旋转效果。注意：当目标为文本类型时，必须使用嵌入字体
SoundEffect	声音效果，播放一个 MP3 声音
WipeLeft WipeRight WipeUp WipeDown	擦除效果，属于遮罩效果。共 4 种，分别对应不同方向
Zoom	缩放效果，以组件为中心进行缩放

11.2.2 为组件添加行为——执行监听动画

为组件添加行为只需要设定相应的触发器事件即可。为了更好地控制动画效果，一般要重新设置动画效果的参数，达到自己期望的效果。

下面创建一个简单的实例来说明组件行为的基本使用方法，具体如代码 11.3 所示。

代码 11.3 组件行为的基本使用方法：EffectTest.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute" backgroundGradientAlphas="[1.0, 1.0]" backgroundGradient-
Colors="[#F26B6B, #FEFCFC]" height="337">
    <mx:Script>
        <![CDATA[
            //动画开始播放
            internal function effectStartHandler():void
            {
                btnEffect.label="动画开始";
            }

            //动画播放结束
```

```
            internal function effectEndHandler():void
            {
                btnEffect.label="动画结束";
            }
        ]]>

    </mx:Script>

    <mx:Iris id= "Iris1" effectStart="effectStartHandler()" effectEnd=
    "effectEndHandler()" />
    <mx:Image id="image" x="100" y="100"  source="@Embed(source='../images/
    Nokia_6630.png')" showEffect="{Iris1}"
    hideEffect="{Iris1}"/>
    <mx:Button label="显示/隐藏"  x="100" y="50" click="image.visible =
    !image.visible" id="btnEffect" fontSize="12"/>
</mx:Application>
```

在代码 11.3 里使用了数据绑定，将 Iris 动画组件和 Image 组件的触发器事件绑在一起，当触发器对应的事件被激发时，动画自动开始播放。MXML 语句中，image 组件的 showEffect 和 hideEffect 都指向了 Iris1。这样，只要 visible 属性发生改变，myIris 动画就会开始播放。Iris1 是 Iris 类型，程序中还监听了它的 effectStart 和 effectEnd 事件。

在 EffectTest.mxml 文件的代码编写完成后保存，在对应项目中右击该文件名称，运行应用程序，效果如图 11-1 所示。

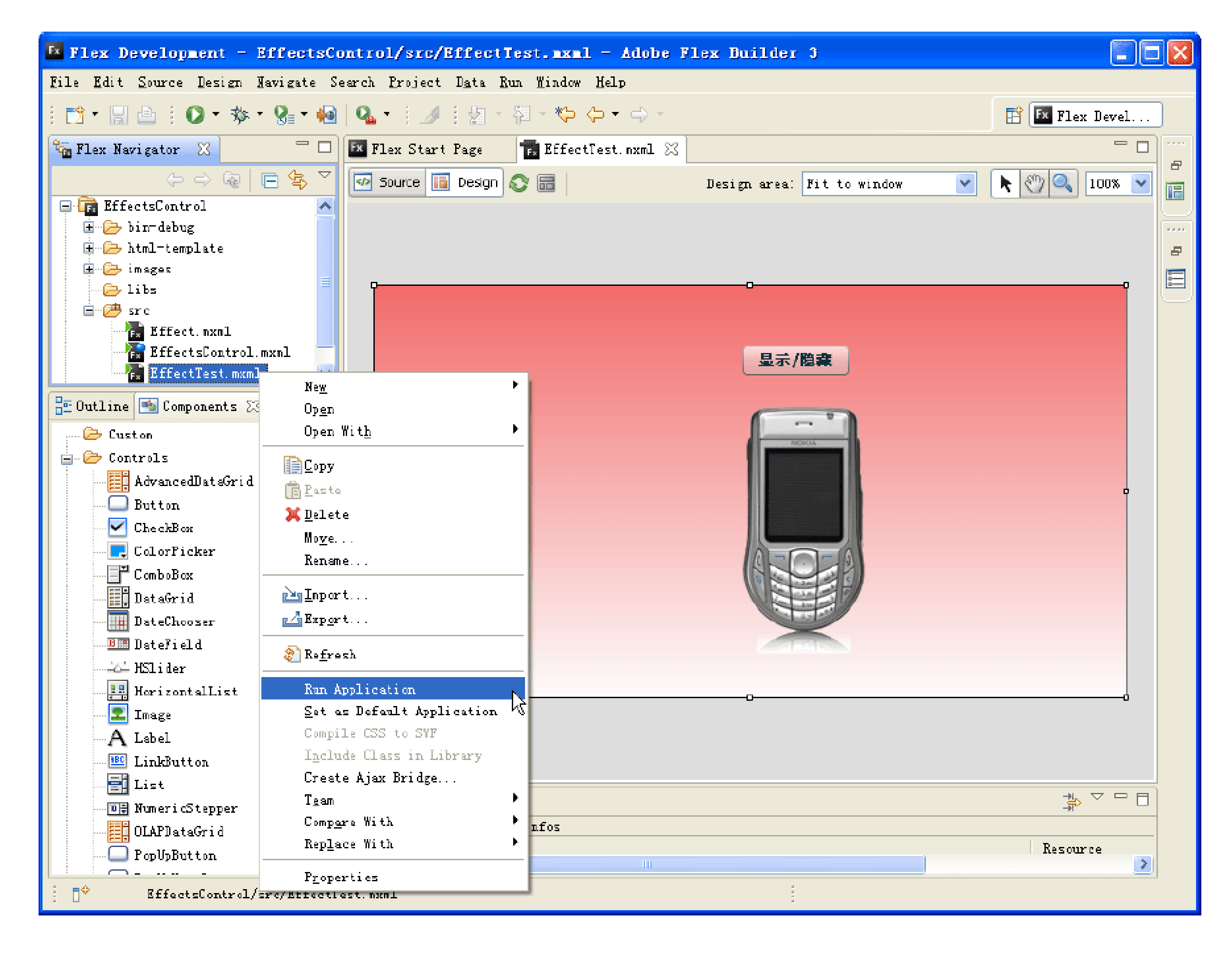


图 11-1　运行应用程序

当应用程序启动后，网页初始化完成，最终效果如图 11-2 所示。

单击页面中的【显示/隐藏】按钮，动画开始执行，按钮上显示“动画开始”，效果如图 11-3 所示。直到动画播放结束，按钮上显示“动画结束”。



图 11-2　网页最终效果

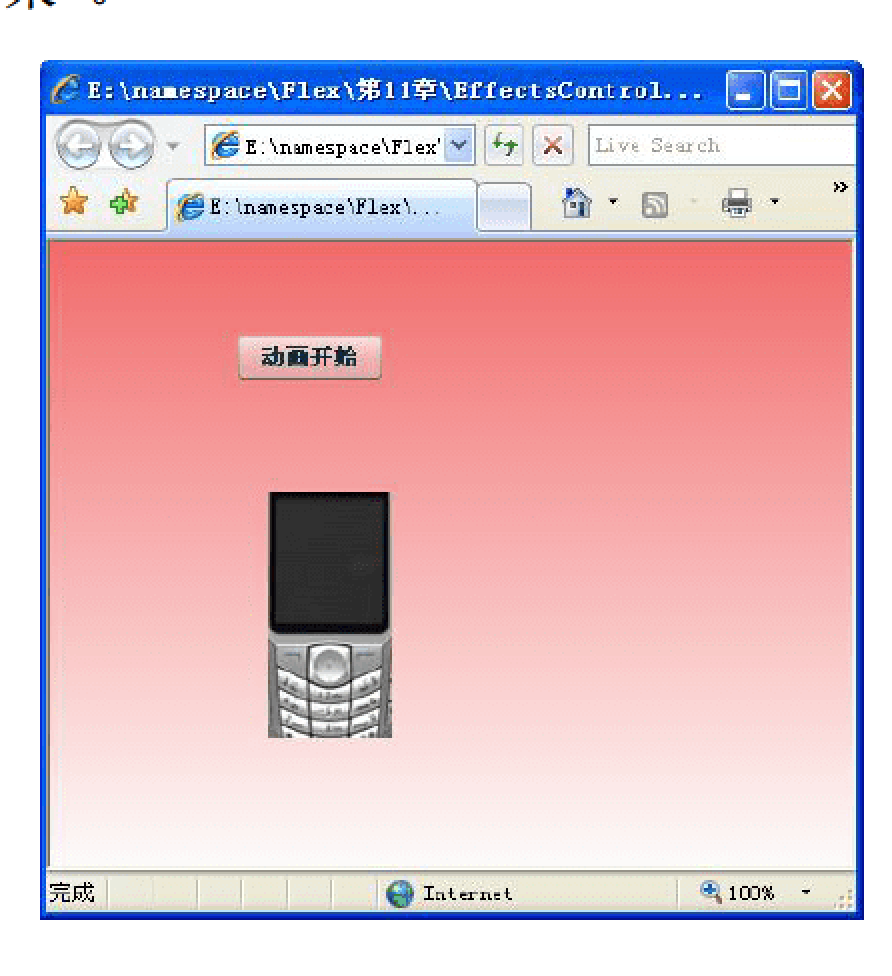


图 11-3　动画开始

11.3　常见动画效果

在上一节中介绍了特效类组件及其运行方式，那么本节将对这些特效组件分别进行详细介绍，包括 Blur（模糊）、Glow（发光）、Fade（淡入淡出）、Dissolve（溶解）和 Iris（彩虹）等。

11.3.1　模糊效果

Blur 组件实现模糊效果，可以让组件变得模糊不清。下面通过一个简单的实例来说明该组件的使用，具体步骤如下所示。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 Blur.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添加一个 Image 组件用于加载一张手机图片并加载两个触发器，添加一个 Text 组件用于显示提示信息，具体如代码 11.4 所示。

代码 11.4　布局图像模糊窗体：Blur.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
    <mx:Panel title="模糊特效实例" width="100%" height="75%"
```

```
            paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
            "10" fontStyle="normal" fontSize="12" fontWeight="bold">

            <mx:Text width="100%" color="#ED5520"
                text="在图像上单击并按住鼠标左键将看到图像模糊效果。松开鼠标后将还原为原图
                 像效果。"/>

            <mx:Image id="flex" source="@Embed(source='../images/Nokia_
            6630.png')"
                mouseDownEffect="{blurImage}"
                mouseUpEffect="{unblurImage}"/>

        </mx:Panel>
    </mx:Application>
```

（3）添加实现图像模糊的 Blur 组件，该组件的 duration 属性设置持续时间，BlurXFrom 和 BlurXTo 设置图片 x 轴上的偏移度，BlurYFrom 和 BlurYTo 设置图片在 y 轴上的偏移度，具体如代码 11.5 所示。

代码 11.5　添加实现图像模糊的 Blur 组件

```
<mx:Blur id="blurImage" duration="1000"
    blurXFrom="0.0" blurXTo="10.0"
    blurYFrom="0.0" blurYTo="10.0"/>
<mx:Blur id="unblurImage" duration="1000"
    blurXFrom="10.0" blurXTo="0.0"
    blurYFrom="10.0" blurYTo="0.0"/>
```

（4）在以上代码添加完成后，使用 Blur 组件制作模糊特效功能已实现。将文件保存后，运行应用程序，页面初始化后的效果如图 11-4 所示。

（5）单击或按下页面中的手机图片，图片将呈现模糊状态，效果如图 11-5 所示。该状态直到鼠标离开 1 秒后结束。



图 11-4　页面初始化后的效果图



图 11-5　图片呈现模糊状态

11.3.2　淡入淡出效果

Fade 组件实现淡入淡出效果，可以让组件变得模糊不清。下面通过一个简单的实例来说明该组件的使用，具体步骤如下所示。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 Fade.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添加一个 Image 组件用于加载一张手机图片并加载两个触发器，添加一个 Text 组件用于显示提示信息，还添加一个 Label 组件用于显示图片名称也加载两个触发器并且与 Image 组件加载的相同，最后添加一个 CheckBox 组件用于控制淡入淡出的状态。由于使用淡入淡出效果以显示或隐藏文字时，需要嵌入字体，因此需使用代码嵌入字体。具体如代码 11.6 所示。

代码 11.6　布局淡入淡出效果窗体：Fade.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute"initialize="Font.registerFont(myriad_font);">
   <mx:Script>
      <![CDATA[

         import flash.text.Font;

         [Embed("../images/Web.ttf", fontName="Myttf")]
         public var myriad_font:Class;
      ]]>
   </mx:Script>

    <mx:Panel title="淡入淡出特效实例" width="100%" height="75%"
      paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
      "10" fontStyle="normal" fontSize="12" fontWeight="bold">

      <mx:Text width="100%" color="#ED5520"
         text="使用淡入淡出效果，以显示或隐藏文字和图片。使用时，嵌入字体采用淡入淡出
         效果的文字。"/>

      <mx:Label text="Nokia 9930"
         fontFamily="Myttf" fontSize="14"
         visible="{chkShow.selected}"
         hideEffect="{fadeOut}" showEffect="{fadeIn}"/>

      <mx:Image source="@Embed(source='../images/Nokia_6630.png')"
         visible="{chkShow.selected}"
```

```
            hideEffect="{fadeOut}" showEffect="{fadeIn}"/>

         <mx:CheckBox id="chkShow" label="visible" selected="true"/>

      </mx:Panel>
   </mx:Application>
```


（3）添加实现淡入淡出效果的 Fade 组件，该组件的 duration 属性设置持续时间，alphaFrom 和 alphaTo 设置图片透明度，1.0 表示不透明，0.0 表示完全透明，具体如代码 11.7 所示。

代码 11.7　添加实现淡入淡出效果的 Fade 组件

```
<mx:Fade id="fadeOut" duration="1000" alphaFrom="1.0" alphaTo="0.0"/>
<mx:Fade id="fadeIn" duration="1000" alphaFrom="0.0" alphaTo="1.0"/>
```

（4）在以上代码添加完成后，使用 Fade 组件制作淡入淡出特效功能已实现。将文件保存后，运行应用程序，页面初始化后的效果如图 11-6 所示。

（5）初始化状态下，visible 复选框处于被选状态，单击该复选框使其处于未选中状态时，图片将呈现淡入状态，效果如图 11-7 所示。当再次单击选中复选框时将呈现淡出状态。



图 11-6　页面初始化后的效果图



图 11-7　图片呈现淡入状态

11.3.3　发光效果

Glow 组件实现发光效果时，使用了 GlowFilter 滤镜。当对组件使用了该效果，不可再使用 GlowFilter 滤镜和其他发光效果。下面通过一个简单的实例来说明该组件的使用，具体步骤如下所示。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一名为 Glow.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添加

一个 Image 组件用于加载一张手机图片并加载两个触发器，添加一个 Text 组件用于显示提示信息，具体如代码 11.8 所示。

代码 11.8　布局图像模糊窗体：Glow.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">

  <mx:Panel title="发光特效实例" width="100%" height="75%"
      paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
      "10" fontStyle="normal" fontSize="12" fontWeight="bold">

      <mx:Text width="100%" color="#ED5520"
         text="单击并按住鼠标时将看到图片发光效果。松开鼠标看到反向发光效果。"/>

      <mx:Image source="@Embed(source='../images/Nokia_6630.png')"
         mouseDownEffect="{glowImage}"
         mouseUpEffect="{unglowImage}"/>

   </mx:Panel>
</mx:Application>
```

（3）添加实现图像发光的 Glow 组件，该组件的 duration 属性设置持续时间，alphaFrom 和 alphaTo 设置图片的透明度，BlurXFrom 和 BlurXTo 及 BlurYFrom 和 BlurYTo 设置图片发光的范围，具体如代码 11.9 所示。

代码 11.9　添加实现图像发光的 Glow 组件

```
<mx:Glow id="glowImage" duration="1000"
   alphaFrom="1.0" alphaTo="0.3"
   blurXFrom="0.0" blurXTo="50.0"
   blurYFrom="0.0" blurYTo="50.0"
   color="0x00FF00"/>
<mx:Glow id="unglowImage" duration="1000"
   alphaFrom="0.3" alphaTo="1.0"
   blurXFrom="50.0" blurXTo="0.0"
   blurYFrom="50.0" blurYTo="0.0"
   color="0x0000FF"/>
```

（4）在以上代码添加完成后，使用 Glow 组件制作发光特效功能已实现。将文件保存后，运行应用程序，页面初始化后的效果如图 11-8 所示。

（5）单击或按下页面中的手机图片，图片将呈现发光状态，效果如图 11-9 所示。该状态直到鼠标离开 1 秒后结束。



图 11-8 页面初始化后的效果图



图 11-9 图片呈现模糊状态

11.3.4 彩虹效果

彩虹效果已经在前面的实例中碰到过，需要由 Iris 组件实现，组件以矩形方式，从中心放大，或缩小到中心，属于遮罩效果。下面通过一个简单的实例来说明该组件的使用，具体步骤如下所示。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 Iris.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添加一个 Image 组件用于加载一张手机图片并加载两个触发器，添加一个 Text 组件用于显示提示信息，添加一个 CheckBox 组件用于控制图片的显示状态。具体如代码 11.10 所示。

代码 11.10 布局图像模糊窗体：Glow.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
  <mx:Panel title="彩虹特效实例" width="100%" height="75%"
      paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
      "10" fontStyle="normal" fontSize="12" fontWeight="bold">

      <mx:Text width="100%" color="#ED5520"
         text="使用彩虹效果能够显示或隐藏图片。"/>

      <mx:Image source="@Embed(source='../images/Nokia 6630.png')"
         visible="{ChkShow.selected}" showEffect="{irisIn}" hideEffect=
         "{irisOut}"/>

      <mx:CheckBox id="ChkShow" label="visible" selected="true"/>
```

```
    </mx:Panel>
</mx:Application>
```

（3）添加实现彩虹特效的 Iris 组件，该组件的 duration 属性设置持续时间，showTarget 属性设置图片的方向，即由内向外或由外向内，具体如代码 11.11 所示。

代码 11.11　添加实现彩虹效果的 Iris 组件

```
<mx:Iris id="irisOut" duration="1000" showTarget="true"/>
<mx:Iris id="irisIn" duration="1000" showTarget="false"/>
```

（4）在以上代码添加完成后，使用 Iris 组件制作彩虹特效功能已实现。将文件保存后，运行应用程序，页面初始化后的效果如图 11-10 所示。

（5）初始化状态下，visible 复选框处于被选中状态，单击该复选框使其处于未选中状态时，图片将呈现由外向内彩虹效果。效果如图 11-7 所示。当再次单击选中复选框时将呈现由内向外彩虹效果。



图 11-10　页面初始化后的效果图

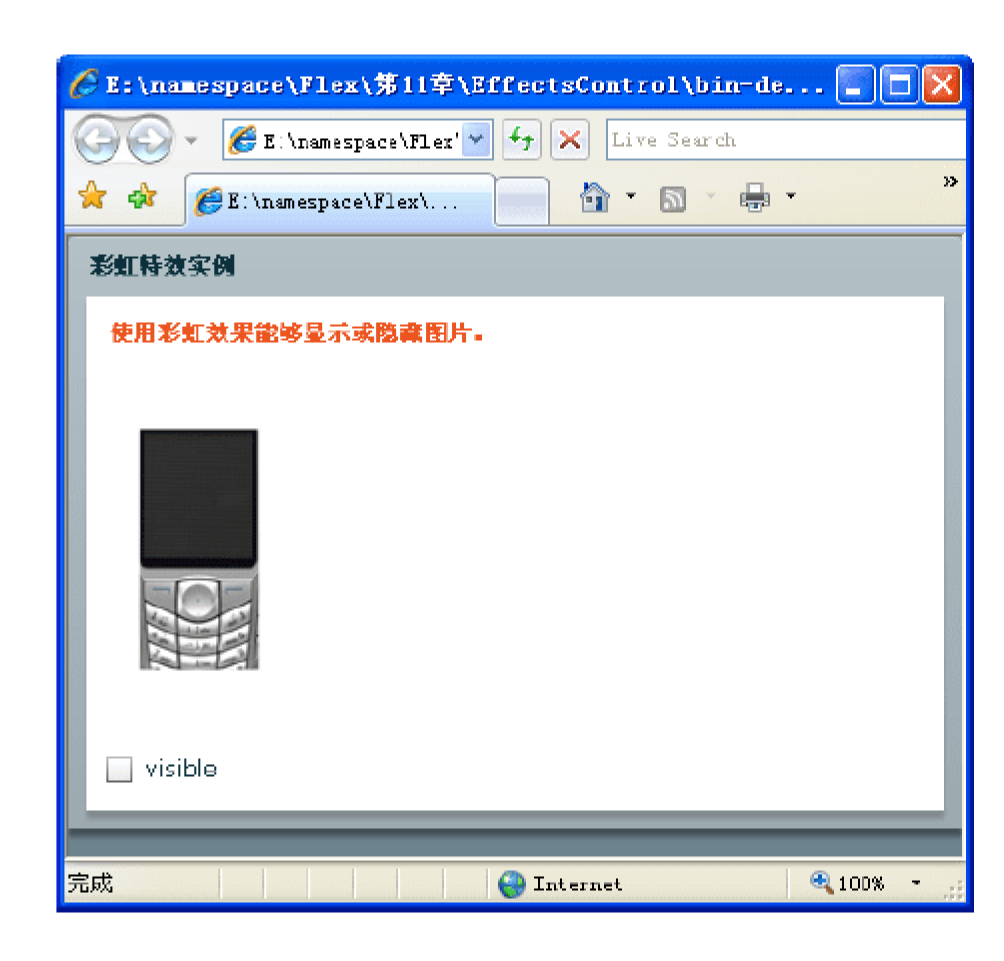


图 11-11　图片呈现由外向内彩虹特效

11.3.5　溶解效果

Dissolve 组件实现溶解效果，主要是在目标对象上增加覆盖层，改变覆盖层的透明度，达到让目标消失或出现的效果。和 Fade 效果相比，该组件可以设置覆盖层的颜色。与 Fade 相同的是当目标对象中包括文字时，必须使用嵌入字体。下面通过一个简单的实例来说明该组件的使用，具体步骤如下所示。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 Dissolve.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添加一个 Image 组件用于加载一张手机图片并加载两个触发器，添加一个 Text 组件用于显示提示信息，还添加一个 Label 组件用于显示图片名称也加载两个触发器并且与 Image 组件加载的相

同，最后添加一个 CheckBox 组件用于控制图片的显示状态。在该实例中还有一个 Button 按钮，所加载的触发器与 Image 完全相同，具体如代码 11.12 所示。

代码 11.12　布局溶解效果窗体：Dissolve.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
    <mx:Panel title="溶解特效实例" width="100%" height="75%"
        paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
        "10" fontStyle="normal" fontSize="12" fontWeight="bold">

        <mx:VBox height="100%">
            <mx:Label text="Nokia 9930"
                fontSize="14"
                visible="{chkShow.selected}"
                hideEffect="{dissolveOut}" showEffect="{dissolveIn}"/>

            <mx:Image source="@Embed(source='../images/Nokia_6630.png')"
                visible="{chkShow.selected}"
                hideEffect="{dissolveOut}" showEffect="{dissolveIn}"/>
        </mx:VBox>

        <mx:VBox height="100%" width="100%">
            <mx:Text width="100%" color="#ED5520"
                text="使用溶解效果显示或隐藏的文字，图片和按钮。"/>

                <mx:Spacer height="100%"/>
                <mx:Button label="Purchase" visible="{chkShow.selected}"
                    hideEffect="{dissolveOut}" showEffect="{dissolveIn}"/>
        </mx:VBox>
        <mx:ControlBar>
            <mx:CheckBox id="chkShow" label="visible" selected="true"/>
        </mx:ControlBar>
    </mx:Panel>
</mx:Application>
```

（3）添加实现溶解效果的 Dissolve 组件，该组件的 duration 属性设置持续时间，alphaFrom 和 alphaTo 属性设置图片透明度，1.0 表示不透明，0.0 表示完全透明，具体如代码 11.13 所示。

代码 11.13　添加实现溶解效果的 Dissolve 组件

```
<mx:Dissolve id="dissolveOut" duration="1000" alphaFrom="1.0" alphaTo=
"0.0"/>
<mx:Dissolve id="dissolveIn" duration="1000" alphaFrom="0.0" alphaTo=
"1.0"/>
```

（4）在以上代码添加完成后，使用 Dissolve 组件制作溶解特效功能已实现。将文件保存后，运行应用程序，页面初始化后的效果如图 11-12 所示。

（5）初始化状态下，visible 复选框处于被选中状态，单击该复选框使其处于未选中状态时，图片、图片标题及按钮都将呈现溶解状态，效果如图 11-13 所示。

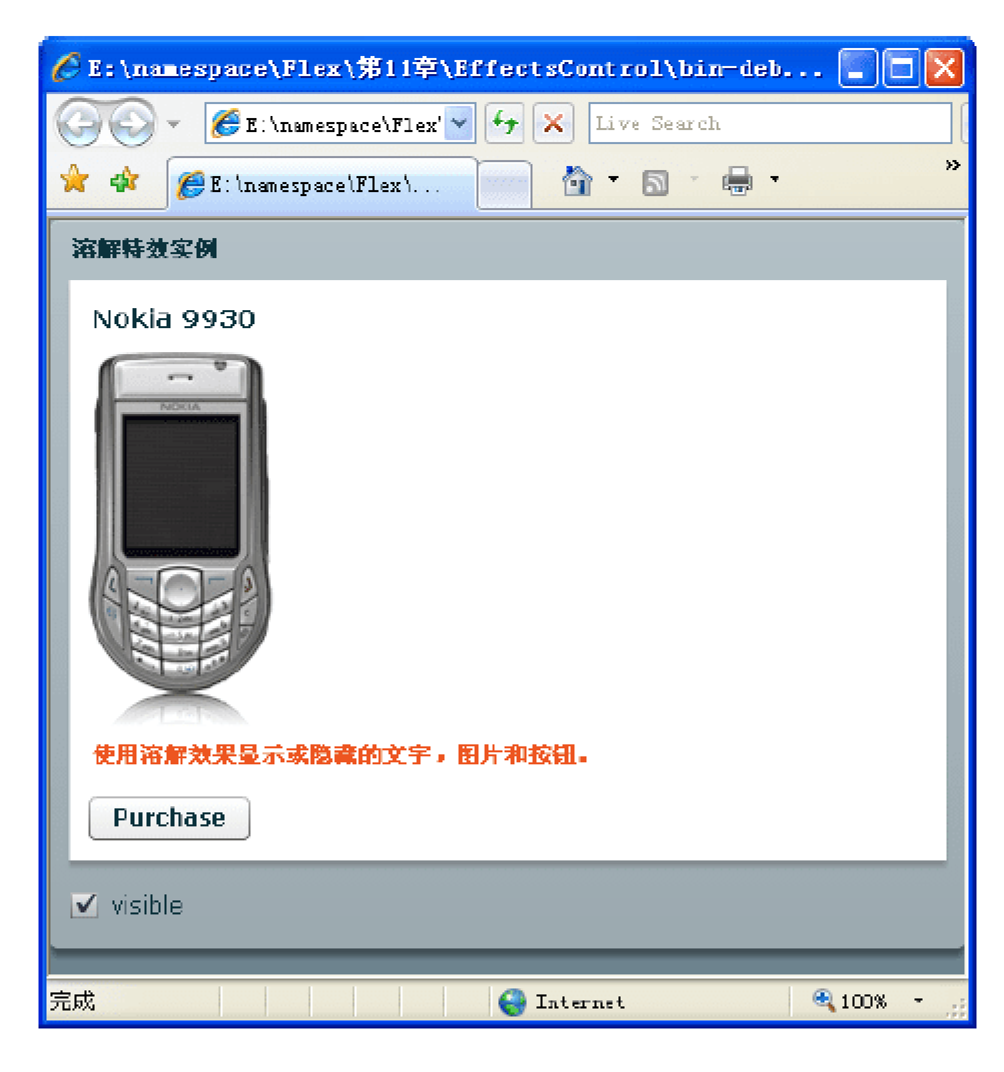


图 11-12　页面初始化后的效果图



图 11-13　呈现溶解状态

11.3.6　移动效果

Move 组件实现移动效果。该组件的坐标，只有当组件位于支持绝对定位的容器中时才有效，如 Canvas、Application、Panel 等。下面通过一个简单的实例来说明该组件的使用，具体步骤如下所示。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 Move.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添加一个 Canvas 组件、一个 Text 组件用于显示提示信息。最后在已经添加的 Canvas 组件中添加一个 Image 组件用于加载一张手机图片，具体如代码 11.14 所示。

代码 11.14　布局移动效果窗体：Move.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
   <mx:Panel title="移动特效实例" width="100%" height="75%"
      paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
      "10" fontStyle="normal" fontSize="12" fontWeight="bold">
     <mx:Text width="100%" color="#ED5520"
        text="在 canvas 容器中的任何位置单击，目标内容将被移动到指定的位置"/>
      <mx:Canvas id="canvas" width="100%" mouseDown="moved();">
         <mx:Image id="img" source="@Embed(source='../images/Nokia_
```

```
            6630.png')" />
        </mx:Canvas>
    </mx:Panel>
</mx:Application>
```

（3）添加实现移动效果的 Move 组件，该组件的 target 属性用于指定目标控件。还要添加当单击 Canvas 组件的任意位置时执行的事件代码，具体如代码 11.15 所示。

代码 11.15　添加实现移动效果的 Move 组件

```
<mx:Script>
    <![CDATA[

        private function moved():void {
            Move1.end();
            Move1.xTo=mouseX-50;
            Move1.play();
        }
  ]]>
</mx:Script>

<mx:Move id="Move1" target="{img}"/>
```

（4）在以上代码添加完成后，使用 Move 组件制作移动特效功能已实现。将文件保存后，运行应用程序，页面初始化后的效果如图 11-14 所示。

（5）初始化状态下，单击 Canvas 容器的任意位置，图片将呈现移动状态，效果如图 11-15 所示。

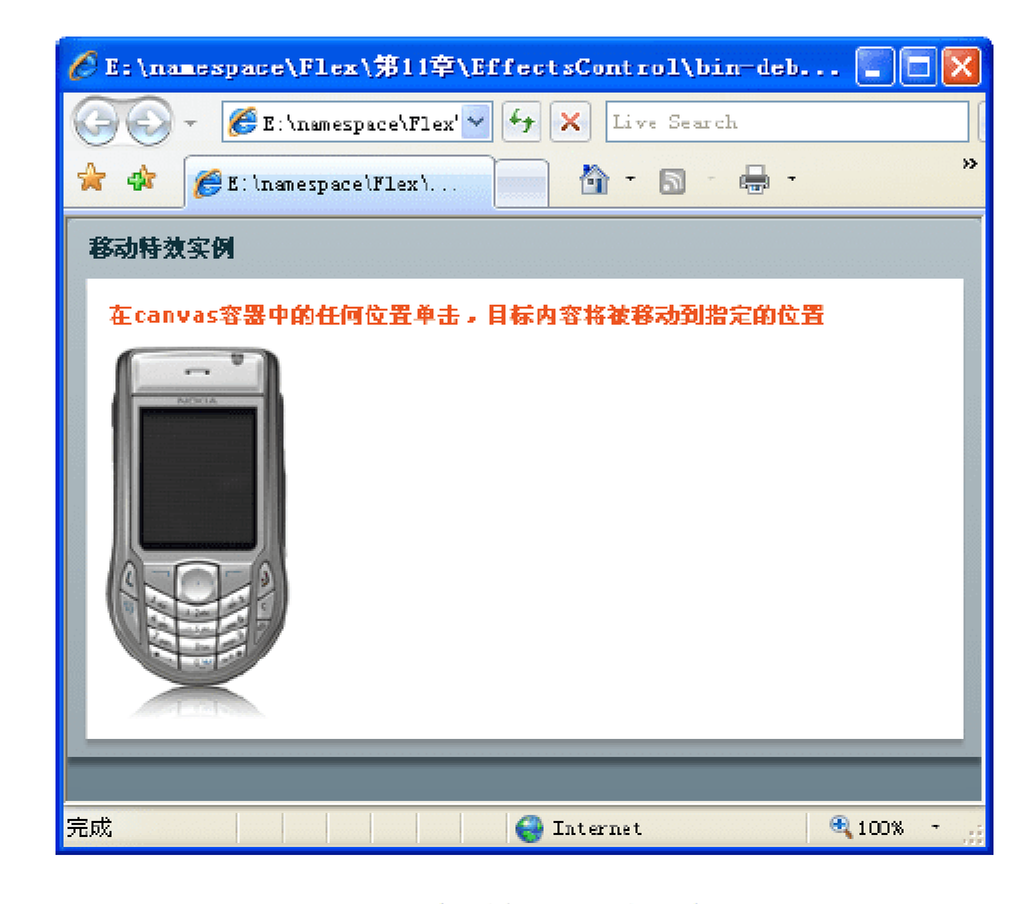


图 11-14　页面初始化后的效果图

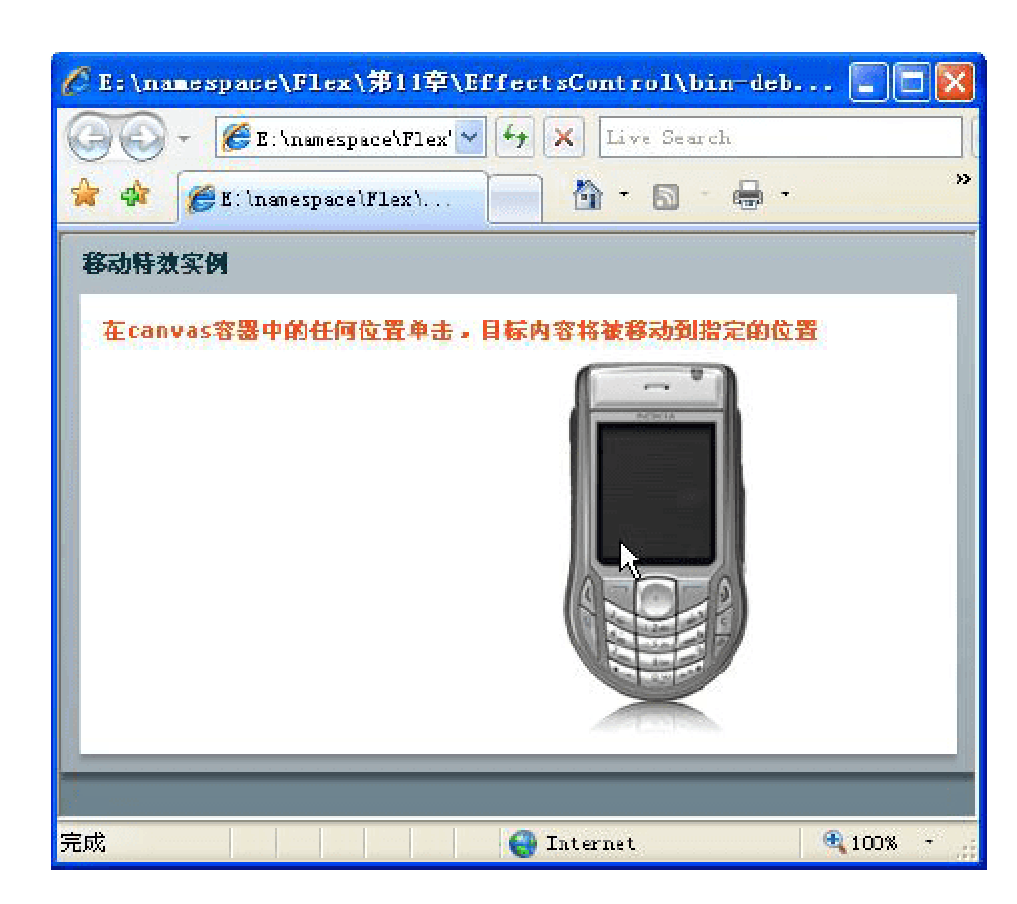


图 11-15　呈现移动效果

11.3.7　尺寸调整效果

Resize 组件实现尺寸调整效果，改变组件的长和宽。当改变组件的长和宽时，处于同一

个容器中的其他组件的大小也可能会相应改变，如果该容器使用了绝对定位则不会发生这种情况。下面通过一个简单的实例来说明该组件的使用，具体步骤如下所示。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 Resize.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添加一个 Image 组件用于加载一张手机图片，添加一个 Text 组件用于显示提示信息，最后再添加两个 Button 组件用于执行放大或缩小操作，具体如代码 11.16 所示。

代码 11.16　布局图像尺寸调整窗体：Resize.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
    <mx:Panel title="尺寸调整特效实例" width="100%" height="75%"
      paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
      "10" fontStyle="normal" fontSize="12" fontWeight="bold">

      <mx:Text width="100%" color="#ED5520" text="使用按钮组件实现图片尺寸调整
      效果"/>

      <mx:Image id="img" source="@Embed(source='../images/Nokia_
      6630.png')" />

      <mx:ControlBar>
         <mx:Button label="放大" click="expand.end(); expand.play();"/>
         <mx:Button label="缩小" click="contract.end(); contract.
         play();"/>
      </mx:ControlBar>

   </mx:Panel>
</mx:Application>
```

（3）添加实现尺寸调整特效的 Resize 组件，该组件的 target 属性设置缩放目标，widthTo 属性设置目标尺寸调整的宽度，heightTo 属性设置目标尺寸调整的高度，具体如代码 11.17 所示。

代码 11.17　添加实现尺寸调整效果的 Resize 组件

```
<mx:Resize id="expand" target="{img}" widthTo="100" heightTo="200"/>
<mx:Resize id="contract" target="{img}" widthTo="50" heightTo="100"/>
```

（4）在以上代码添加完成后，使用 Resize 组件制作尺寸调整特效功能已实现。将文件保存后，运行应用程序，页面初始化后的效果如图 11-16 所示。

（5）初始化状态下，目标图片宽 100、高 200，与单击放大效果时设置的相同，因此，单击【放大】按钮将无任何效果，而单击【缩小】按钮效果如图 11-17 所示。当再次单击【放大】

按钮时将返回到原始效果。



图 11-16　页面初始化后的效果图

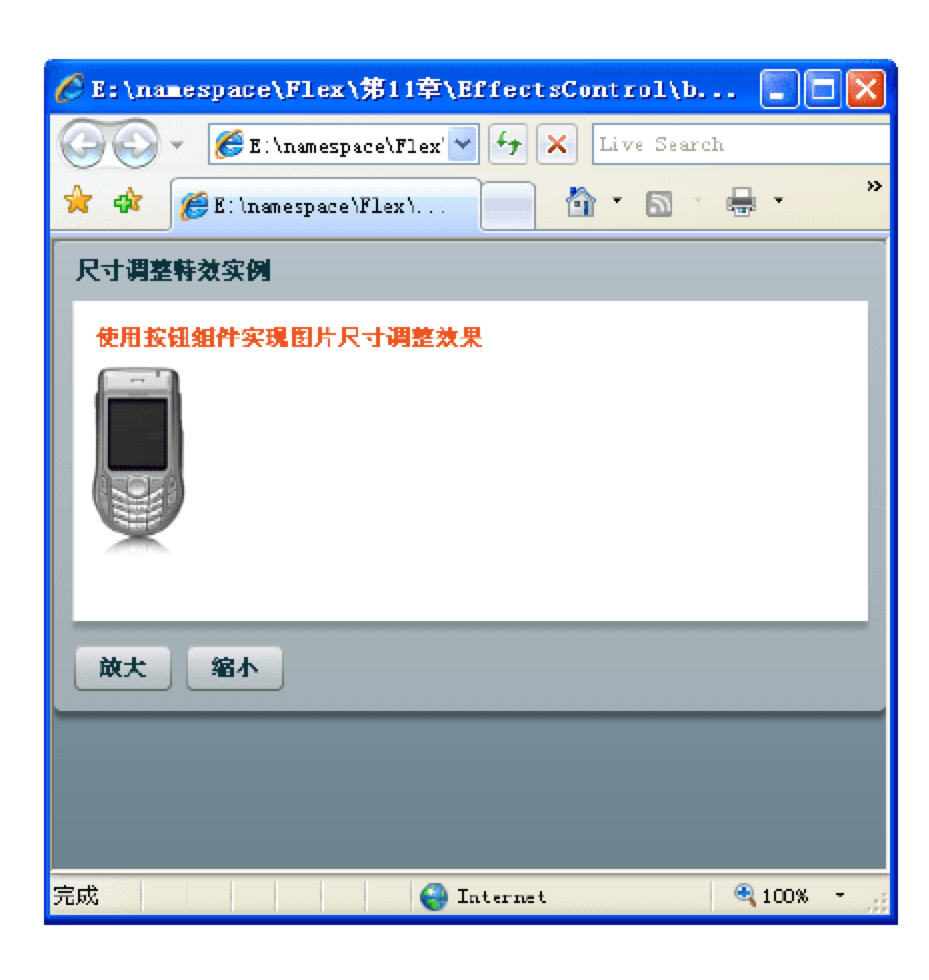


图 11-17　单击“缩小”按钮时效果

11.3.8　旋转效果

Rotate 组件实现旋转效果，可以让指定目标按照指定的某一角度执行旋转。下面通过一个简单的实例来说明该组件的使用，具体步骤如下所示。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 Rotate.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添加一个 Image 组件用于加载一张手机图片，添加一个 Text 组件用于显示提示信息，还添加一个 Label 用于显示图片名称。具体如代码 11.18 所示。

代码 11.18　布局旋转效果窗体：Rotate.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute"
   initialize="Font.registerFont(myweb_font);">
   <mx:Panel title="缩放特效实例" width="100%" height="75%"
      paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
      "10" fontStyle="normal" fontSize="12" fontWeight="bold">

      <mx:VBox id="myVB" width="50%" horizontalAlign="center">
         <mx:Label text="Nokia 9930" fontFamily="MyWeb" fontSize="14"/>
         <mx:Image id="img" source="@Embed(source='../images/Nokia_
         6630.png')" />
      </mx:VBox>

       <mx:Text width="100%" color="#ED5520"
```

```
            text="单击按钮，图片旋转 30 度. 当目标为文本类型时，必须使用嵌入字体。"/>

        <mx:ControlBar>
            <mx:Button label="旋转 30 度" click="rotated();"/>
        </mx:ControlBar>
    </mx:Panel>
</mx:Application>
```

（3）添加实现旋转效果的 Rotate 组件，该组件的 angleFrom 属性设置从哪个角度开始，angleTo 属性设置到哪个角度。由于目标中包含文本类型，需要嵌入字体。具体如代码 11.19 所示。

代码 11.19　添加实现旋转效果的 Rotate 组件

```
<mx:Script>
    <![CDATA[

        import flash.text.Font;

        [Embed("../images/Web.ttf", fontName="MyWeb")]
        public var myweb_font:Class;

        [Bindable]
        public var angle:int=0;

        private function rotated():void {
            rotate.end();
            angle += 30;
            rotate.play();
        }
  ]]>
</mx:Script>

<mx:Rotate id="rotate" angleFrom="{angle-30}" angleTo="{angle}" target=
"{myVB}"/>
```

（4）在以上代码添加完成后，使用 Rotate 组件制作旋转特效功能已实现。将文件保存后，运行应用程序，页面初始化后的效果如图 11-18 所示。

（5）初始化状态下，单击【旋转 30 度】按钮，图片将旋转 30 度，效果如图 11-19 所示。当再次单击时将从上次旋转位置开始继续以 30 度旋转。

11.3.9　声音效果

SoundEffect 组件实现声音效果，用于播放一个 MP3 声音。下面通过一个简单的实例来说明该组件的使用，具体步骤如下所示。




图 11-18　页面初始化后的效果图



图 11-19　图片旋转后状态

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 Sonnd Effect.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添加一个 Image 组件用于加载一张手机图片，添加一个 Label 组件用于显示提示信息，具体如代码 11.20 所示。

代码 11.20　布局声音特效窗体：SoundEffect.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
   <mx:Panel title="声音特效实例" width="100%" height="75%"
      paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
      "10" fontStyle="normal" fontSize="12" fontWeight="bold">

      <mx:Label width="100%" color="#ED5520"
         text="单击图片将听到 MP3 音乐。"/>

      <mx:Image id="img" source="@Embed(source='../images/Nokia_
      6630.png')"
         mouseDownEffect="{mySounds}"/>

   </mx:Panel>
</mx:Application>
```

（3）添加实现声音特效的 SoundEffect 组件，该组件的 source 属性设置文件来源，具体如代码 11.21 所示。

代码 11.21　添加实现声音特效的 SoundEffect 组件

```
   <mx:SoundEffect id="mySounds" source="@Embed(source='../images/
```

```
qq.mp3')"/>
```

（4）在以上代码添加完成后，使用 SoundEffect 组件制作声音特效功能已实现。将文件保存后，运行应用程序，页面初始化后的效果如图 11-20 所示。

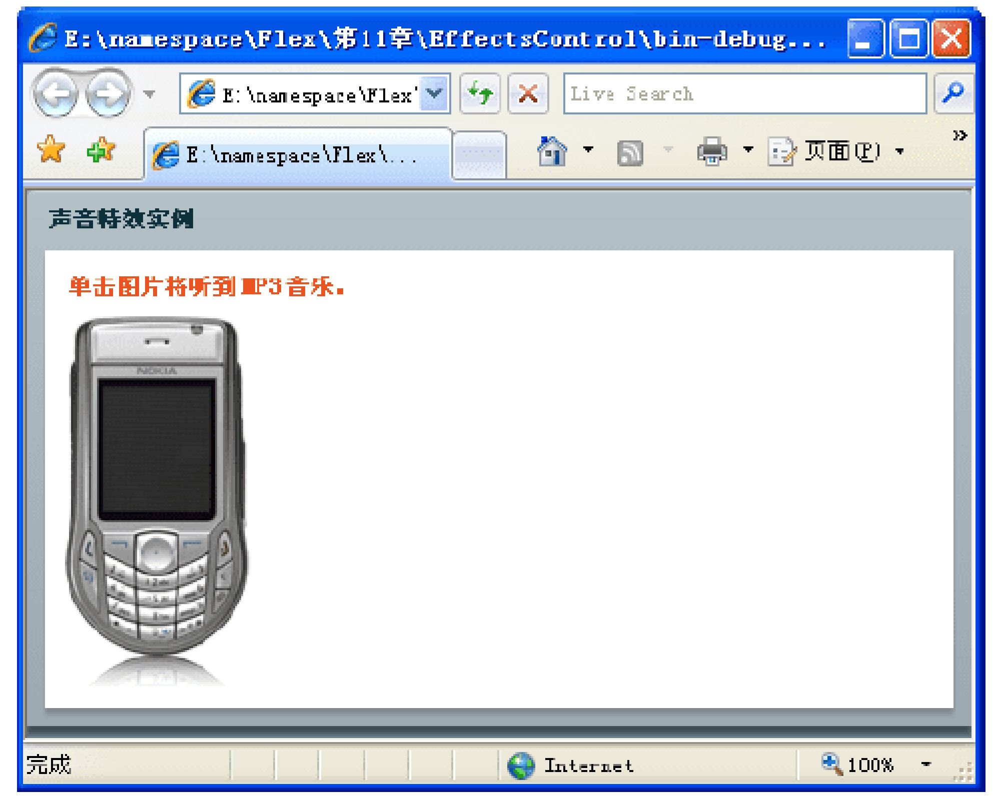


图 11-20　页面初始化后的效果图

（5）单击页面中的手机图片，将听到很短的一段声音，不管你播放的 MP3 有多大、多长，这里就只能听到很少的一部分。因此，在这里建议使用体积尽可能小的 MP3 播放。

11.3.10　缩放效果

Zoom 组件实现缩放效果，以组件为中心进行放缩。下面通过一个简单的实例来说明该组件的使用，具体步骤如下所示。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 ZoomEffect.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添加一个 Image 组件用于加载一张手机图片并且加载鼠标进入和移出时执行的事件，添加一个 Text 组件用于显示提示信息。具体如代码 11.22 所示。

代码 11.22　布局图像尺寸调整窗体：ZoomEffect.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
    <mx:Panel title="缩放特效实例" width="100%" height="75%"
        paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
        "10" fontStyle="normal" fontSize="12" fontWeight="bold">

        <mx:Text width="100%" color="#ED5520"
            text="将鼠标移动到目标位置内容上时将放大图片，如果鼠标离开目标位置内容将缩小
            图片"/>

        <mx:Image id="img"
            source="@Embed(source='../images/Nokia_6630.png')"
            scaleX=".5" scaleY=".5"
            rollOver="doZoom(event)"
            rollOut="doZoom(event)"/>

    </mx:Panel>
</mx:Application>
```

（3）添加实现缩放效果的 Zoom 组件，该组件的 zoomWidthTo 属性设置目标缩放的宽度，

ZoomHeiqhtTo 属性设置目标缩放的高度，具体如代码 11.23 所示。

代码 11.23　添加实现缩放效果的 Zoom 组件

```
<mx:Script>
    <![CDATA[
        import flash.events.MouseEvent;
        public function doZoom(event:MouseEvent):void {
            if (zoomAll.isPlaying) {
                zoomAll.reverse();
            }
            else {
                zoomAll.play([event.target], event.type == MouseEvent.ROLL
                OUT ? true : false);
            }
        }
    ]]>
</mx:Script>

<mx:Zoom id="zoomAll" zoomWidthTo="1" zoomHeightTo="1" zoomWidthFrom=".5"
zoomHeightFrom=".5"  />
```

（4）在以上代码添加完成后，使用 Zoom 组件制作缩放效果功能已实现。将文件保存后，运行应用程序，页面初始化后的效果如图 11-21 所示。

（5）初始化状态下，将鼠标移动到图片任意位置上时效果如图 11-22 所示。当离开图片时将返回到原始效果。



图 11-21　页面初始化后的效果图



图 11-22　鼠标移动到图片任意位置上时效果

11.3.11　擦除效果

擦除效果共有 WipeLeft、WipeRight、WipeUp 和 WipeDown4 种，分别对应不同方向，属

于遮罩效果。这 4 种方式的工作机制基本相同，只是方向不同，这里仅以 WipeUp 组件实现擦除效果为例进行说明。下面通过一个简单的实例来说明该组件的使用，具体步骤如下所示。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 WipeUp.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添加一个 Image 组件用于加载一张手机图片并加载两个触发器，添加一个 Text 组件用于显示提示信息，还添加一个 Label 组件用于显示图片名称也加载两个触发器并且与 Image 组件加载的相同。最后添加一个 CheckBox 组件用于控制淡入淡出的状态，具体如代码 11.24 所示。

代码 11.24　布局图像擦除效果窗体：WipeUp.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
   <mx:Panel title="擦除特效实例" width="100%" height="75%"
      paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
      "10" fontStyle="normal" fontSize="12" fontWeight="bold">

      <mx:Text width="100%" color="#ED5520"
         text="使用 WipeUp 特效组件能够实现文本或图像的擦除。"/>

      <mx:Label text="Nokia 9930"
         fontSize="14"
         visible="{chkShow.selected}"
         hideEffect="{wipeOut}" showEffect="{wipeIn}"/>
      <mx:Image source="@Embed(source='../images/Nokia_6630.png')"
         visible="{chkShow.selected}"
         hideEffect="{wipeOut}" showEffect="{wipeIn}"/>

      <mx:CheckBox id="chkShow" label="visible" selected="true"/>

   </mx:Panel>
</mx:Application>
```

（3）添加实现擦除效果的 WipeUp 组件，该组件的 duration 属性设置擦除持续时间，具体如代码 11.25 所示。

代码 11.25　添加实现擦除效果的 WipeUp 组件

```
<mx:WipeUp id="wipeOut" duration="1000"/>
<mx:WipeUp id="wipeIn" duration="1000"/>
```

（4）在以上代码添加完成后，使用 WipeUp 组件制作擦除特效功能已实现。将文件保存后，运行应用程序，页面初始化后的效果如图 11-23 所示。

（5）初始化状态下，visible 复选框处于被选中状态，单击该复选框使其处于未选中状态

时，图片将呈现擦除状态，效果如图 11-24 所示。当再次单击选中复选框时将呈现显示状态。




图 11-23　页面初始化后的效果图



图 11-24　呈现擦除效果

11.3.12　复合效果

所谓复合效果就是将多个动画效果组合到一块，同时执行的效果。这里以使用 Parallel 组件实现复合效果为例进行说明。下面通过一个简单的实例来说明该组件的使用，具体步骤如下所示。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 Parallel.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添加一个 Image 组件用于加载一张手机图片，添加一个 Text 组件用于显示提示信息，还添加两个 Button 组件用于执行相关操作，具体如代码 11.26 所示。

代码 11.26　布局图像复合效果窗体：Parallel.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"absolute">
    <mx:Panel title="复合效果实例" width="100%" height="75%"
        paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
        "10">

        <mx:Text width="100%" color="#ED5520"
            text="使用按钮组件移动图片位置并更改图片大小。"/>

        <mx:Canvas id="canvas" width="100%" height="100%">
            <mx:Image id="img" x="20" y="20" width="30" height="60"
                source="@Embed(source='../images/Nokia_6630.png')"/>
```

```
        </mx:Canvas>

        <mx:ControlBar>
            <mx:Button label="播放" click="expand.end(); expand.play();"/>
            <mx:Button label="返回" click="contract.end(); contract.
            play();"/>
        </mx:ControlBar>

    </mx:Panel>
</mx:Application>
```

（3）添加实现复合特效的 Parallel 组件，该复合操作由 Move 和 Resize 两个组件来完成。Paralle 组件的 target 属性设置操作目标，具体如代码 11.27 所示。

代码 11.27　添加实现复合效果的 Parallel 组件

```
<mx:Parallel id="expand" target="{img}">
    <mx:Move xTo="{canvas.width/2 - 50}" yTo="{canvas.height/2 - 100}"/>
    <mx:Resize widthTo="100" heightTo="200"/>
</mx:Parallel>

<mx:Parallel id="contract" target="{img}">
    <mx:Move xTo="20" yTo="20"/>
    <mx:Resize widthTo="30" heightTo="60"/>
</mx:Parallel>
```

（4）在以上代码添加完成后，使用 Parallel 组件制作复合特效功能已实现。将文件保存后，运行应用程序，页面初始化后的效果如图 11-25 所示。

（5）初始化状态下，单击【播放】按钮图片将一边移动，一边放大，直到指定值为止，效果如图 11-26 所示。单击【返回】按钮时，将执行逆向操作。

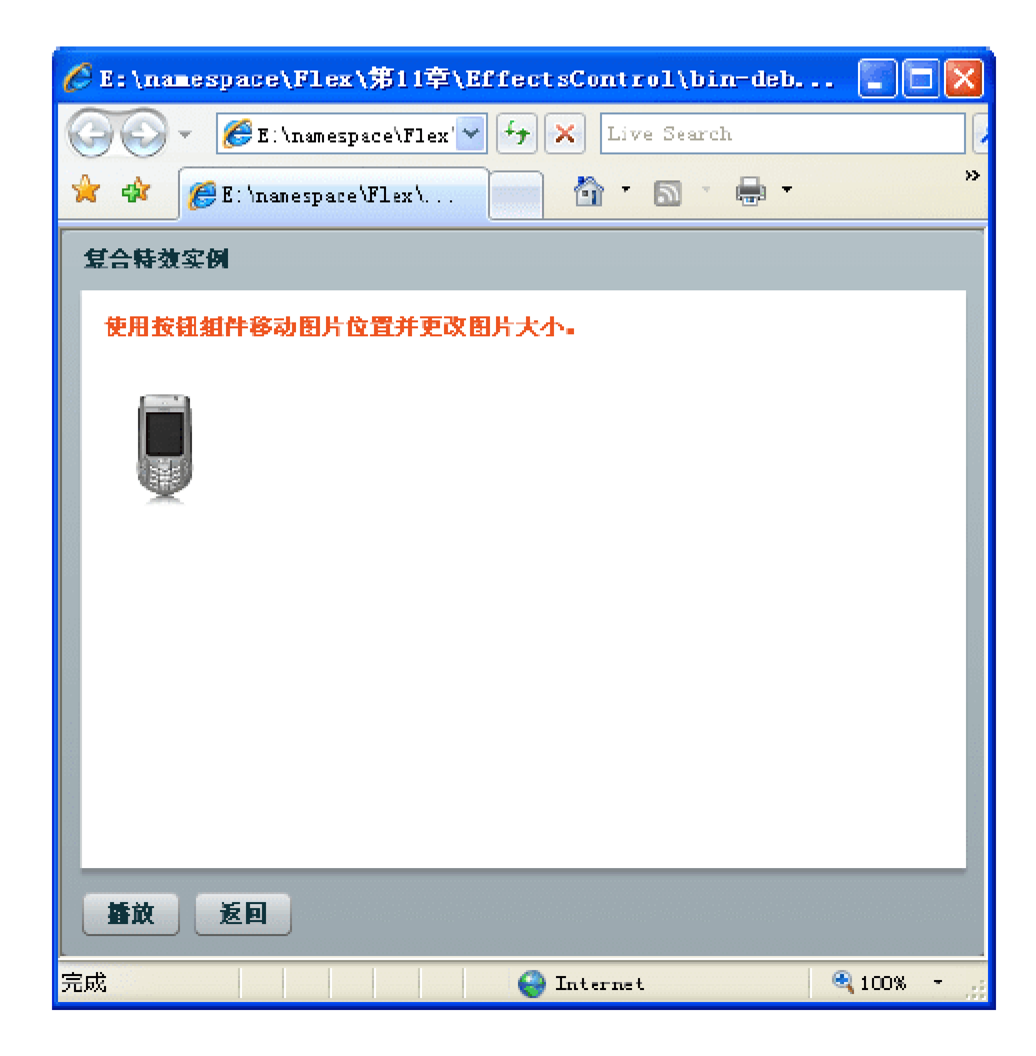


图 11-25　页面初始化后的效果图



图 11-26　最终效果

11.4 行为和状态

在 Adobe Flex 中使用状态和变换可以创建更为丰富、更具互动性的用户体验。例如，可以使用状态去创建用户界面，根据用户所执行的操作来改变它的外观。

11.4.1 使用 State 对象

简单地说，状态定义组件的某个特定视图。例如，产品缩略图可以有两个视图状态：包含次要信息的基本状态和包含附加信息的富状态。相似地，应用程序可以有与不同应用程序状况相对应的多个视图状态，如登录状态、注册状态或搜索结果状态。

下面的实例使用状态很容易地实现登录和注册表单。在此实例中，初始视图状态提示用户登录，并根据需要包含让注册的链接。如果用户选择【单击这里注册】链接，则该表单会改变视图状态以显示注册信息。当用户单击【返回登录】链接时，视图状态会变回到登录表单。如果单击【下一步】按钮则将进入填写详细信息的状态。下面看一下该实例的具体步骤。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 States.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 Panel 组件，然后在该组件中添一个 Text 组件用于显示提示信息，添加两个 Form 组件用作输入用户名和密码的文本框，添加两个 Button 组件分别用于登录与注册等，具体如代码 11.28 所示。

代码 11.28　布局登录表单：States.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
backgroundGradientAlphas="[1.0, 1.0]" backgroundGradientColors="[#F32B2B,
#FDFAFA]">
    <!--使用 Panel 容器制作一个用户登录状态-->
    <mx:Panel title="用户登录" id="loginPanel"
        horizontalScrollPolicy="off" verticalScrollPolicy="off"
        paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom="10"
        fontSize="12" width="316">

        <mx:Text width="100%" color="#FF5400"
 text="如果您还不是本站用户，请单击【单击这里注册】按钮，注册一个新用户。"id="text1"/>

        <mx:Form id="loginForm" >
            <mx:FormItem label="用户名:">
                <mx:TextInput/>
            </mx:FormItem>
            <mx:FormItem label="密   码:" id="formitem1">
```

```
                <mx:TextInput/>
            </mx:FormItem>
        </mx:Form>
        <mx:ControlBar id="controlbar1">
            <mx:LinkButton id="returnLogin"  label="单击这里注册"
                click="currentState='Register'"/>
            <mx:Spacer width="100%" id="spacer1"/>
            <mx:Button label="登录" id="Next"/>
        </mx:ControlBar>
    </mx:Panel>
</mx:Application>
```

（3）在以上内容添加完成后，用户登录的表单设计效果已制作完成，效果如图 11-27 所示。



图 11-27　用户登录的表单设计效果

（4）要实现状态转换需要使用 State 对象，因此需要在 States.mxml 文件的 Source 状态下添加 State 对象代码，这里输入代码内容如代码 11.29 所示。

代码 11.29　添加 State 对象代码

```
<mx:states>
    <mx:State name="Register">
    </mx:State>
</mx:states>
```

（5）转换到 States.mxml 文件的 Design 状态，在 State 下拉列表中显示出已经创建的 Register 状态，效果如图 11-28 所示。

（6）通过查看 Register 状态可知，该状态下效果与用户登录表单完全相同。由于用户注册信息的首页仅包含用户名、密码及重复密码的输入。当用户单击【返回登录】链接时，视图状态会变回到登录表单。如果单击【下一步】按钮则将进入填写详细信息的状态。因此只需对 Register 状态用户登录表单进行适当的更改即可。更改完毕后，最终代码如代码 11.30 所示。



图 11-28　Register 状态

代码 11.30　Register 状态最终代码

```
<mx:states>
    <mx:State name="Register">
        <mx:SetProperty target="{loginForm}" name="height" value="119"/>
        <mx:AddChild relativeTo="{loginForm}" position="lastChild">
            <mx:FormItem label="重复密码:" id="formitem0">
             <mx:TextInput/>
            </mx:FormItem>
        </mx:AddChild>
        <mx:SetProperty target="{Next}" name="label" value="下一步"/>
        <mx:SetProperty target="{returnLogin}" name="label" value="返回登录"/>
            <mx:SetProperty target="{formitem1}" name="label" value="密　码:"/>
            <mx:SetProperty target="{text1}" name="text" value="请认真填写您的注
            册信息"/>
        <mx:SetEventHandler target="{returnLogin}" name="click" handler=
        "currentState=''"/>
        <mx:SetEventHandler target="{Return}" name="click" handler=
        "currentState='Details'"/>
    </mx:State>
</mx:states>
```

（7）在 Register 状态下切换到 Design 设计视图，查看用户注册首页最终设计效果。效果如图 11-29 所示。

（8）由于需要填写用户注册的详细信息，因此还需再添加一个状态，这里设置该状态的名称为 Details，具体如代码 11.31 所示。

代码 11.31　用户注册的详细信息

```
<mx:State name="Details">
    <mx:RemoveChild target="{returnLogin}"/>
    <mx:SetProperty target="{Return}" name="label" value="完成"/>
    <mx:SetProperty target="{formitem2}" name="label" value="真实姓名:"/>
```

```
        <mx:SetProperty target="{text1}" name="text" value="请如实填写您的信息，方便我们与你联系。"/>
        <mx:SetProperty target="{formitem1}" name="label" value="性别:"/>
        <mx:SetProperty target="{loginForm}" name="height" value="148"/>
        <mx:AddChild relativeTo="{loginForm}" position="lastChild">
            <mx:FormItem label="手机:" id="formitem3">
                <mx:TextInput/>
            </mx:FormItem>
        </mx:AddChild>
        <mx:AddChild relativeTo="{loginForm}" position="lastChild">
            <mx:FormItem label="Email:" id="formitem4">
                <mx:TextInput/>
            </mx:FormItem>
        </mx:AddChild>
        <mx:SetProperty target="{loginPanel}" name="width" value="358"/>
    </mx:State>
```



图 11-29　用户注册首页设计效果

（9）在 Details 状态下切换到 Design 设计视图，查看用户注册详细信息的最终设计效果。效果如图 11-30 所示。



图 11-30　用户注册详细信息设计效果

（10）在用户登录、用户注册首页及用户注册详细信息 3 种状态添加完成后，使用 State 对象制作用户登录的实例已制作完毕。将文件保存后，运行应用程序，页面初始化后的效果如图 11-31 所示。

（11）单击【单击这里注册】按钮，进入用户注册表单，效果如图 11-32 所示。




图 11-31　页面首次运行效果

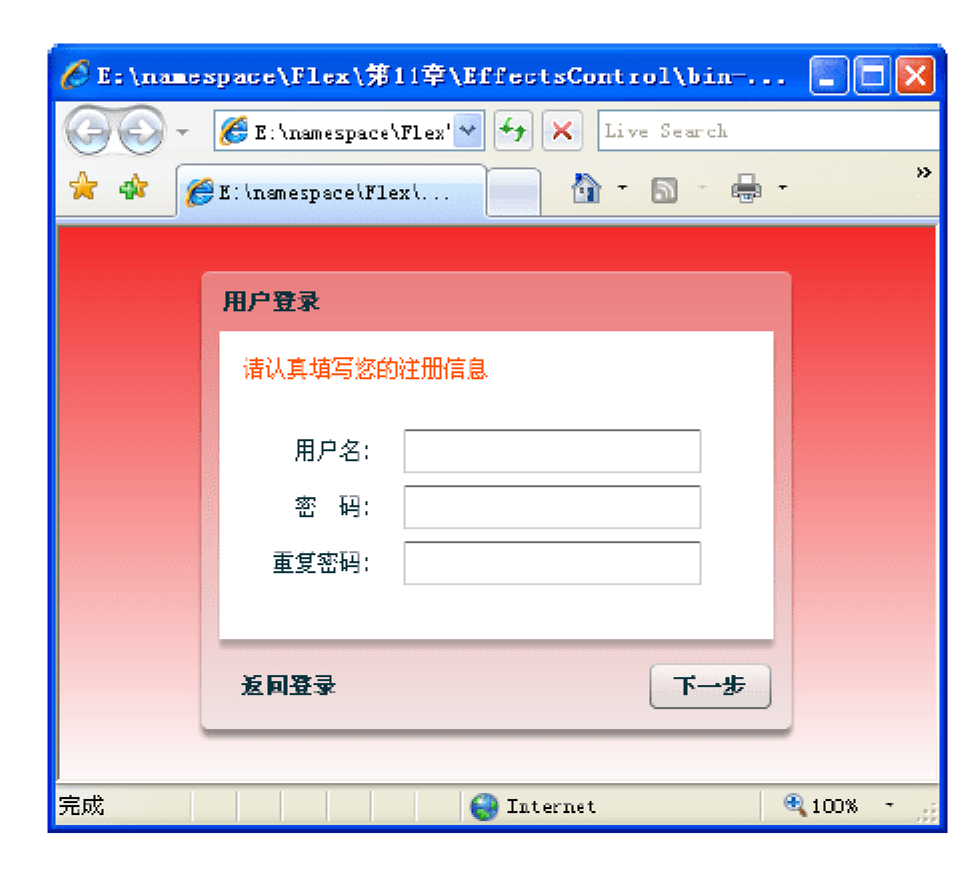


图 11-32　用户注册表单

（12）在用户注册状态下，单击【返回登录】按钮将返回登录状态，单击【下一步】按钮将进入 Details 状态，继续填写用户详细信息，效果如图 11-33 所示。



图 11-33　用户详细信息

11.4.2　使用 Transition 对象

Transition（形变动画）的使用极大地增强了状态模式下界面的表现力。Transition（形变动画）对象位于 mx.states 包中，主要包括两个属性：fromState 和 toState，分别代表过渡动作的前后状态。当状态切换符合 Transition 的条件，即切换前后的状态和 fromState、toState 都相等时，形变动画就开始播放。

下面就以对上一节中用户登录及注册状态之间相互切换为例来说明 Transition 对象的使用

方法。具体步骤如下所示。

（1）打开已经创建完成的 EffectsControl 项目，在 src 目录下添加一个名为 Transition.mxml 的 MXML Application 文件。

（2）Transition.mxml 文件与 States.mxml 文件的布局与状态基本相同，只做了少许的改动，这里不再详细介绍，具体如代码 11.32 所示。

代码 11.32　表单布局：Transition.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical"
backgroundGradientAlphas="[1.0, 1.0]" backgroundGradientColors="[#F32B2B,
#FDFAFA]">
    <mx:states>
        <mx:State name="Register">
            <mx:AddChild relativeTo="{loginForm}" position="lastChild">
                <mx:target>
                    <mx:FormItem id="confirm" label="重复密码:">
                        <mx:TextInput/>
                    </mx:FormItem>
                </mx:target>
            </mx:AddChild>
            <mx:SetProperty target="{loginPanel}" name="title" value="
            用户注册"/>
            <mx:SetProperty target="{loginButton}" name="label" value="注册"/>
            <mx:SetStyle target="{loginButton}"
                name="color" value="#FE0D01"/>
            <mx:RemoveChild target="{registerLink}"/>
            <mx:AddChild relativeTo="{spacer1}" position="before">
                <mx:target>
                    <mx:LinkButton id="loginLink" label="返回登录" click=
                    "currentState=''"/>
                </mx:target>
            </mx:AddChild>
        </mx:State>
     </mx:states>

    <mx:Panel title="用户登录" id="loginPanel"
        horizontalScrollPolicy="off" verticalScrollPolicy="off"
        paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom="10"
        fontSize="12" width="314">

        <mx:Text width="100%" color="blue"
            text="使用 Transition 为状态添加变行动作"/>

        <mx:Form id="loginForm" >
```

```
            <mx:FormItem label="用户名:" id="formitem1">
                <mx:TextInput/>
            </mx:FormItem>
            <mx:FormItem label="密   码:">
                <mx:TextInput/>
            </mx:FormItem>
        </mx:Form>
        <mx:ControlBar>
            <mx:LinkButton id="registerLink"  label="单击这里注册"
                click="currentState='Register'"/>
            <mx:Spacer width="100%" id="spacer1"/>
            <mx:Button label="登录" id="loginButton"/>
        </mx:ControlBar>
    </mx:Panel>
</mx:Application>
```

（3）使用 Transition（形变动画）为转入 Register 状态及默认状态添加形变动作，从而极大地增强状态模式下界面的表现力，具体如代码 11.33 所示。

代码 11.33　使用 Transition 添加形变动作

```
<mx:transitions>
    <mx:Transition id="toRegister" fromState="*" toState="Register">
        <mx:Sequence targets="{[loginPanel, registerLink, confirm, loginLink,
        spacer1]}">
            <mx:RemoveChildAction/>
            <mx:SetPropertyAction target="{loginPanel}" name="title"/>
            <mx:SetPropertyAction target="{loginButton}" name="label"/>
            <mx:SetStyleAction target="{loginButton}" name="color"/>
            <mx:Resize target="{loginPanel}" duration="1000"/>
            <mx:AddChildAction/>
        </mx:Sequence>
    </mx:Transition>

    <mx:Transition id="toDefault" fromState="Register" toState="*">
        <mx:Sequence targets="{[loginPanel, registerLink,
                confirm, loginLink, spacer1]}">
            <mx:RemoveChildAction/>
            <mx:SetPropertyAction target="{loginPanel}" name="title"/>
            <mx:SetPropertyAction  target="{loginButton}" name="label"/>
            <mx:SetStyleAction target="{loginButton}" name="color"/>
            <mx:Resize target="{loginPanel}"/>
            <mx:AddChildAction/>
        </mx:Sequence>
    </mx:Transition>
</mx:transitions>
```

（4）以上内容添加完成后，使用 Transition 对象为状态添加形变动作的实例已制作完毕。将文件保存后，在左侧的列表中右击该文件名，选择 Run Application 命令，运行应用程序，页面初始化后的效果如图 11-34 所示。

（5）单击【单击这里注册】按钮，进入用户注册表单。在切换状态的过程中，将逐步进行，出现平滑的效果，如图 11-35 所示。





图 11-34　页面首次运行效果



图 11-35　切换状态的过程中用户注册表单效果

（6）由于状态切换的时间设置为 1 秒，因此在经过 1 秒后，动作切换完毕，最终效果如图 11-36 所示。

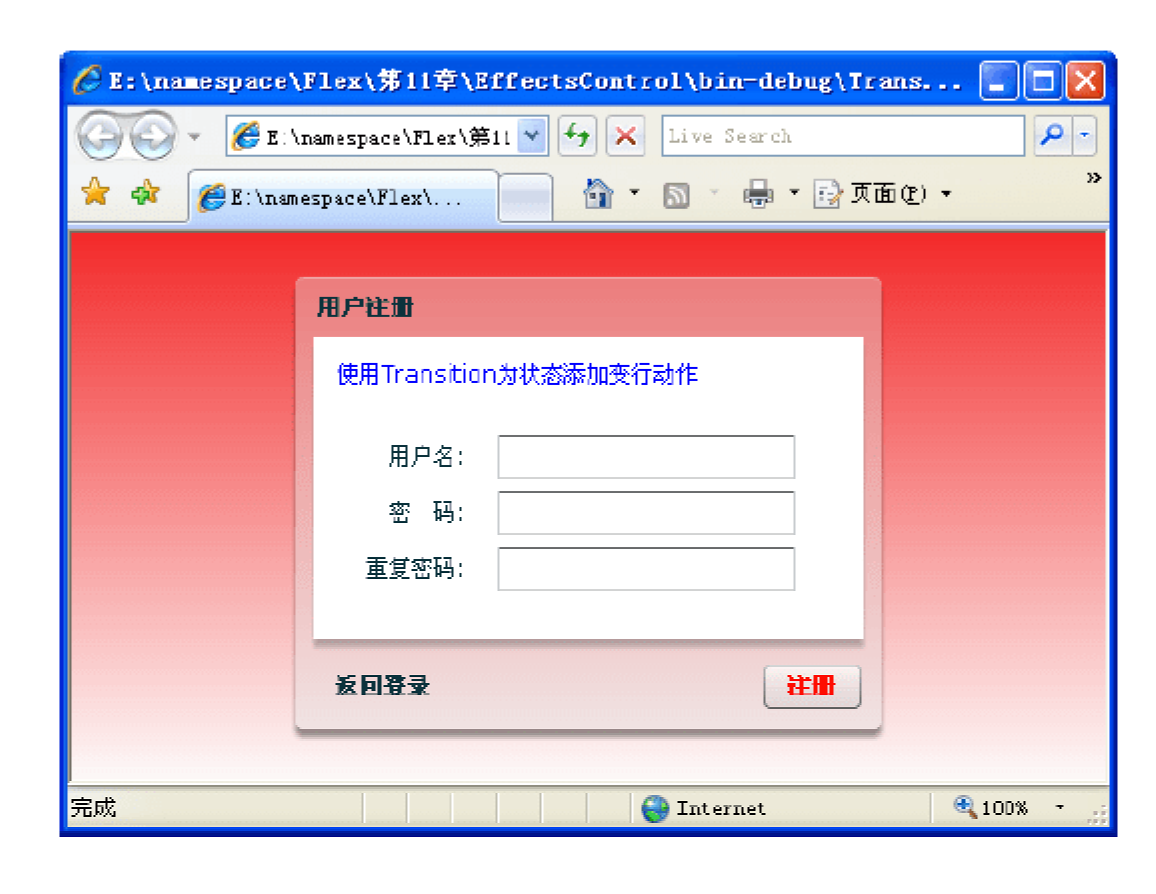
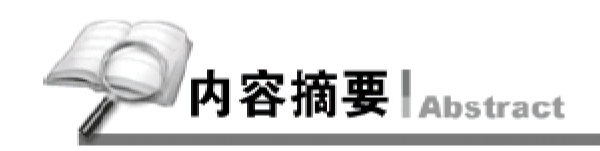


图 11-36　用户注册表单最终效果

第12章 事件机制



内容摘要 Abstract

很多 Flex 的初学者对事件机制都不太熟悉，在使用过程中难免会出现各种问题。这是一个非常普遍的现象。为了更好地使用 Flex，本章将介绍 Flex 中的事件机制和使用方法，重点是使读者理解事件的工作流程、了解 Event 对象并掌握如何使用自定义事件。

Flex 的精髓之一就是事件机制，理解之后，能帮助大家更灵活地设计程序，也对初学者有很大的帮助。



学习目标 Objective

- 理解观查者模式的概念
- 了解 ActionScript 3.0 的可视化对象架构
- 掌握事件流的运行流程
- 掌握 Event 对象的重要属性和方法
- 熟练掌握创建自定义事件的方法
- 掌握事件机制的高级应用

12.1 观查者模式

在介绍事件机制之前，首先弄明白一个概念，什么叫设计模式。设计模式（Designpattern）是一套被反复使用的、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了重用代码、让代码更容易被他人理解、保证代码的可靠性。

观查者模式也是设计模式的一种，该设计模式在 Java 编程中已经得到广泛的应用。那么在 Flex 中，什么是观查者模式呢？例如，很多家庭需要到邮局订阅杂志，那么首先需要到邮局登记注册，以后邮局就会在规定时间里将杂志邮寄到所登记的住址。也就是说事件的使用者首先需要进行注册，注册完毕后无须再关心，一旦有事件需要触发，注册中心将通知事件使用者。

Actionscript 3.0 中使用的就是观查者模式，示例代码如下所示。

```
button.addEventListener("click",onClick);
internal function onClick(Evt:MouseEvent):void{
    Alert.show("单击了一个按钮！","温馨提示");
```

```
    }
```

在上面的代码中，button 是 Button 组件的 id，该按钮调用了一个 addEventListener()方法，添加事件监听者。也就是为按钮注册一个事件，其中字符串“click”表示事件注册的类型，onClick 表示事件发生时如何执行操作。下面是一个事件注册函数，该函数有一个要求，其参数必须是一个事件类型。这里的 MouseEvent 是一个鼠标事件类型。当在按钮上单击时，onClick 事件将被触发。

上面提及到了 addEventListener()方法，该方法以后将应用非常广泛，下面对该方法进行详细的介绍。

```
public function addEventListener(type:String,
    listener:Function,
    useCapture:Boolean=false,
    priority:int=0,
    useWeakReference:Boolean=false
    ):void
```

使用 EventDispatcher 对象注册事件监听器对象，以使监听器能够接收事件通知。可以为特定类型的事件、阶段和优先级在显示列表的所有节点上注册事件监听器。

成功注册一个事件监听器后，无法通过额外调用 addEventListener()方法来更改其优先级。要更改监听器的优先级，必须首先调用 removeListener()方法。然后，可以使用新的优先级再次注册该监听器。

注册该监听器后，如果继续调用具有不同 type 或 useCapture 值的 addEventListener()方法，则会创建单独的监听器注册。例如，如果首先注册 useCapture 设置为 true 的监听器，则该监听器只在捕获阶段进行监听。如果使用同一个监听器对象再次调用 addEventListener()方法，并将 NseCapture 设置为 false，那么便会拥有两个单独的监听器：一个在捕获阶段进行监听，另一个在目标和冒泡阶段进行监听。

不能只为目标阶段或冒泡阶段注册事件监听器。这些阶段在注册期间是成对出现的，因为冒泡阶段只适用于目标节点的始终。

如果不再需要某个事件监听器，可调用 removeEventListener()方法将其删除，否则会产生内存问题。由于垃圾回收器不会删除仍包含引用的对象，因此不会从内存中自动删除使用已注册事件监听器的对象。

复制 EventDispatcher 实例时并不复制其中附加的事件监听器。(如果新创建的节点需要一个事件监听器，必须在创建该节点后附加该监听器)。但是，如果移动 EventDispatcher 实例，则其中附加的事件监听器也会随之移动。

如果在正在处理事件的节点上注册事件监听器，则不会在当前阶段触发事件监听器，但会在事件流的稍后阶段触发，如冒泡阶段。

如果从正在处理事件的节点中删除事件监听器，则该事件监听器仍由当前操作触发。删除事件监听器后，绝不会再次调用该事件监听器（除非再次注册以备将来处理）。

addEventListener()方法非常重要，下面介绍其各参数的具体作用，参数名称及其作用如表 12-1 所示。

表 12-1　addEventListener()方法的参数名称及作用

参数名称	作用
type:String	事件的类型
listener:Function	处理事件的监听器函数。此函数必须接受 Event 对象作为其唯一的参数，并且不能返回任何结果
useCapture:Boolean	确定监听器是运行于捕获阶段、目标阶段还是冒泡阶段。如果将 useCapture 设置为 true，则监听器只在捕获阶段处理事件，而不在目标或冒泡阶段处理事件。如果 useCapture 为 false，则监听器只在目标或冒泡阶段处理事件。要在所有 3 个阶段都监听事件，请调用两次 addEventListener()方法，第一次将 useCapture 设置为 true，第二次再将 useCapture 设置为 false
priority:int	事件监听器的优先级。优先级由一个带符号的 32 位整数指定。数字越大，优先级越高。优先级为 n 的所有监听器会在优先级为 n-1 的监听器之前得到处理。如果两个或更多个监听器共享相同的优先级，则按照它们的添加顺序进行处理。默认优先级为 0
useWeakReference:Boolean	确定对监听器的引用是强引用，还是弱引用。强引用（默认值）可防止用户的监听器被当作垃圾回收。弱引用则没有此作用。类级别成员函数不属于垃圾回收的对象，因此可以将类级别成员函数的 useWeakReference 设置为 true 而不会使它们受垃圾回收的影响。如果将作为嵌套内部函数的监听器的 useWeakReference 设置为 true，则该函数将被作为垃圾回收并且不再是永久函数。如果创建对该内部函数的引用（将该函数保存到另一个变量中），则该函数将不被当作垃圾回收而保持永久

下面看一个关于鼠标注册事件的完整实例，具体步骤如下所示。

（1）创建一个名为 EventDemo 的 Flex 项目，在 src 目录下将自动添加一个名为 EventDemo.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。添加一个 id 为 btnMessage 的 Button 组件用于注册单击事件，添加一个 id 为 canvas1 的组件用于注册鼠标经过时的事件，添加一个 TextArea 组件用于显示提示信息，具体如代码 12.1 所示。

代码 12.1　布局窗体：EventDemo.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete=
"Page_load()" layout="absolute" fontSize="12">
    <mx:Button id="btnMessage" x="24" y="9" label="注册事件" fillAlphas="[1.0,
    1.0]" fillColors="[#F71A1A, #FFFEFE]" width="151"/>
    <mx:Canvas id="canvas1" x="24" y="96" width="292" height="195"
    backgroundColor="#49FA03">
    </mx:Canvas>
    <mx:TextArea id="txt1" x="355" y="10" width="242" height="281"/>

</mx:Application>
```

（3）分别为 btnMessage 按钮注册鼠标单击时触发事件，为 canvas1 画布添加鼠标经过时触发事件，具体如代码 12.2 所示。

代码 12.2　注册鼠标事件

```
<mx:Script>
    <![CDATA[
        import mx.controls.Alert;

        internal function Page_load():void
        {
            btnMessage.addEventListener(MouseEvent.CLICK,onClick);
            canvas1.addEventListener(MouseEvent.MOUSE_OVER,onMouseOver);
        }

    ]]>
</mx:Script>
```

（4）当鼠标单击按钮事件触发时将弹出一个提示对话框，当鼠标经过画布事件触发时将在文本区域中显示提示信息，具体如代码 12.3 所示。

代码 12.3　添加事件触发时执行的操作

```
internal function onClick(Evt:MouseEvent):void{
    Alert.show("单击了一个按钮！","温馨提示");
}
internal function onMouseOver(Evt:MouseEvent):void
{
    ShowMessage("鼠标经过画布",txt1);
}
internal function ShowMessage(str:String,txt:TextArea):void
{
    txt.text+=str+"\n";
        }
```

（5）在以上代码添加完成后，关于鼠标注册事件的实例已制作完成。将文件保存后，运行应用程序，页面初始化后的效果如图 12-1 所示。

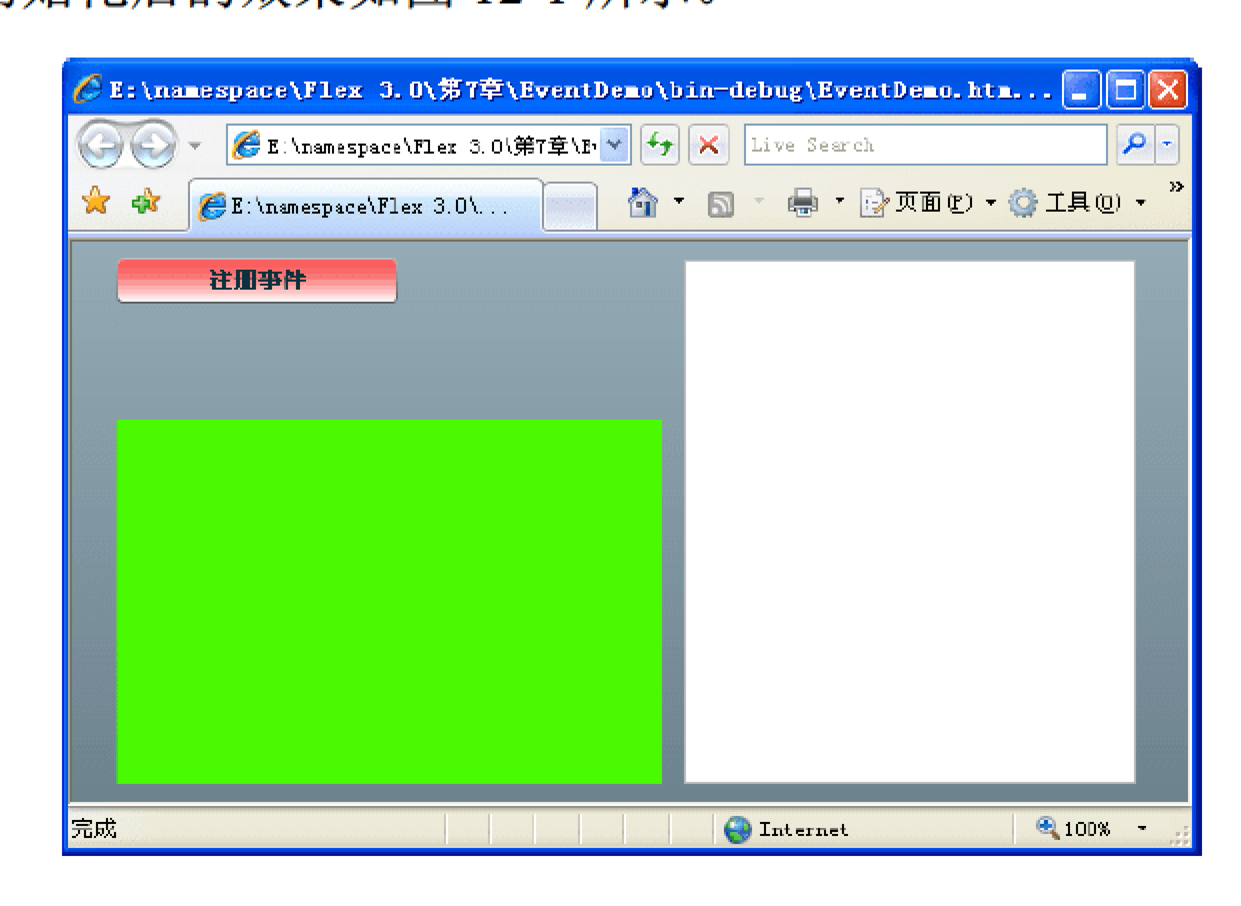


图 12-1　页面初始化后的效果图

（6）当单击【注册事件】按钮时，将弹出【温馨提示】对话框，显示“单击了一个按钮!”，效果如图 12-2 所示。

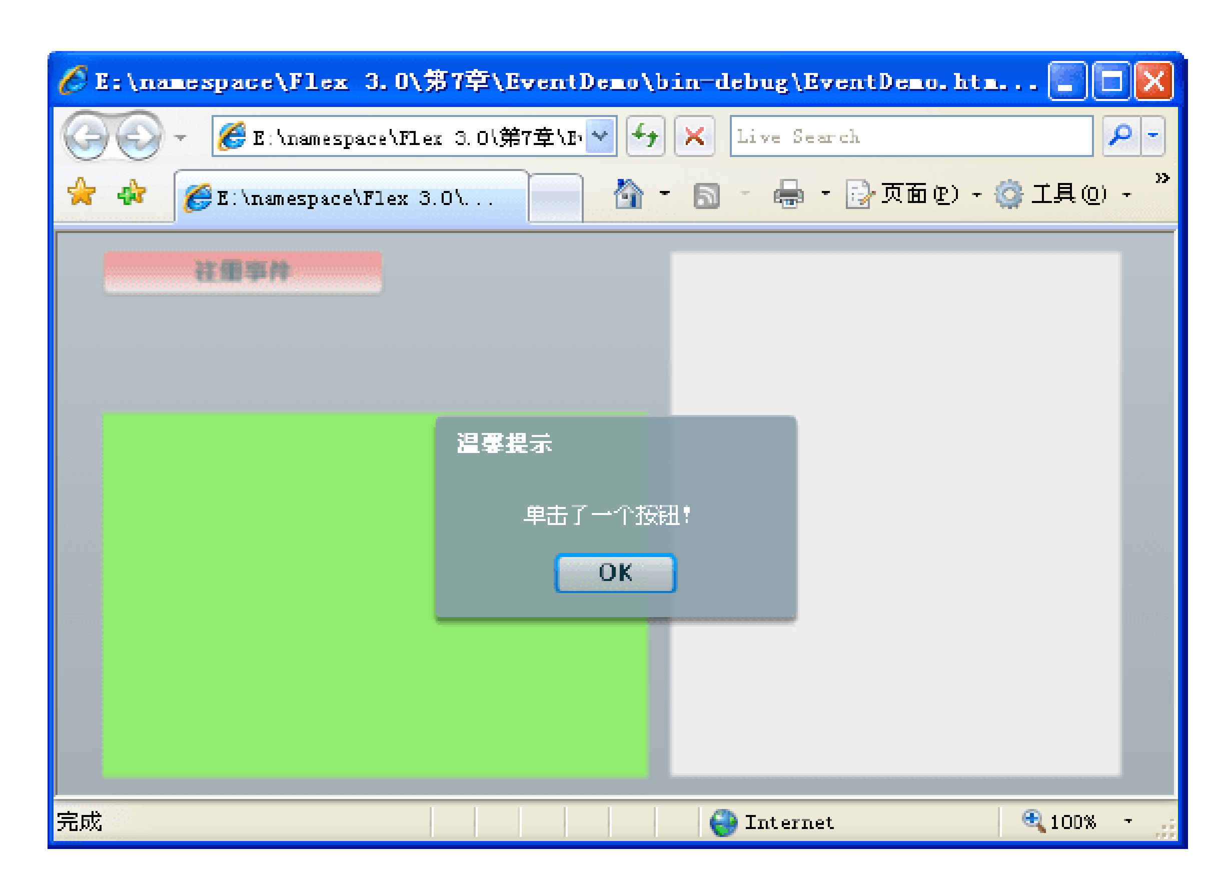


图 12-2　单击【注册事件】按钮

（7）当鼠标每次经过绿色的画布时，文本区域中将显示“鼠标经过画布”，在鼠标经过画布 5 次后，效果如图 12-3 所示。

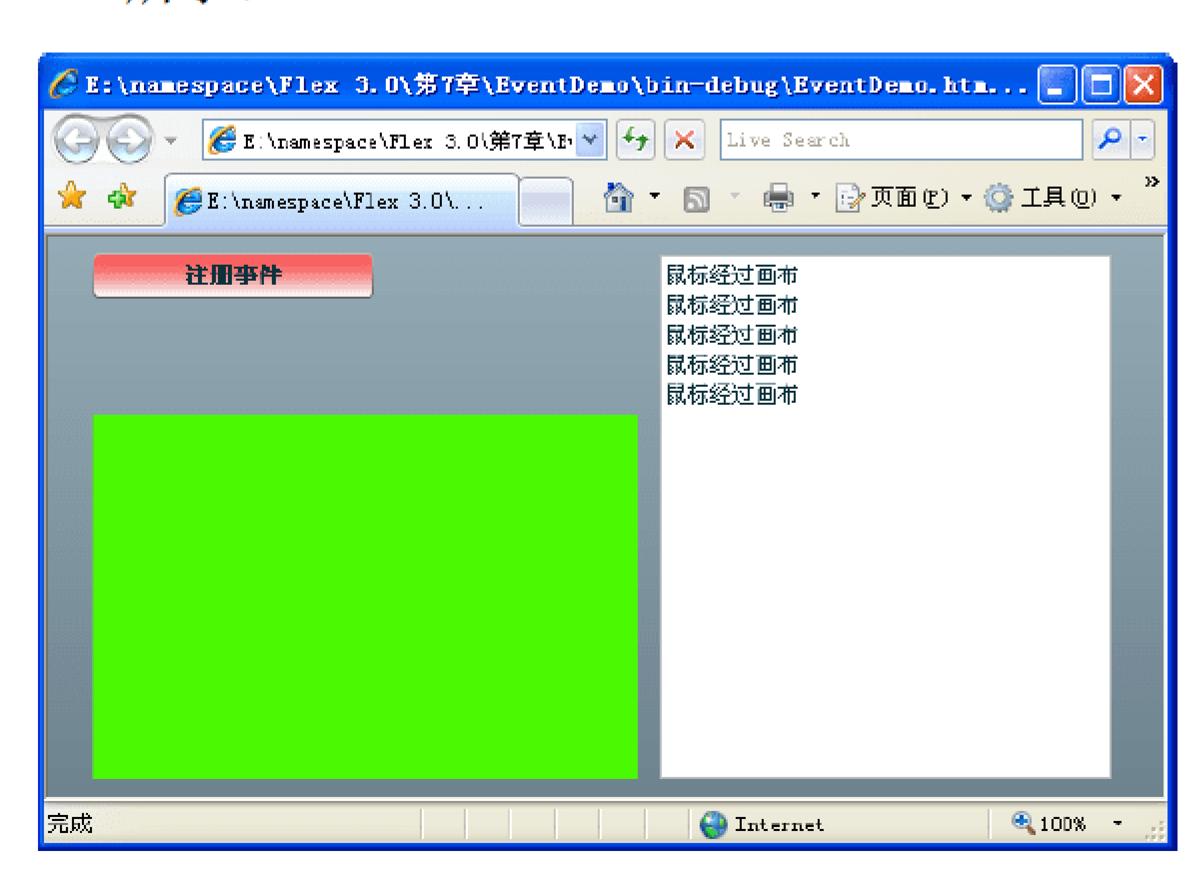


图 12-3　鼠标 5 次经过画布

12.2　ActionScript 3.0 的可视化对象架构

在 ActionScript 2.0 中，对象要具有派发事件的能力，必须经过特殊的功能扩展，否则是无法使用事件模式的，更谈不上事件监听。所以，为了实现事件模型，开发者需要自己动手，对现有对象进行扩展。到了 ActionScript 3.0 中，所有可视化对象已经内建了事件机制。事件机制类的继承关系如图 12-4 所示。

图 12-4 从上到下依次列举了事件机制类的继承关系。下面分别对各个类进行简单介绍，具体如表 12-2 所示。

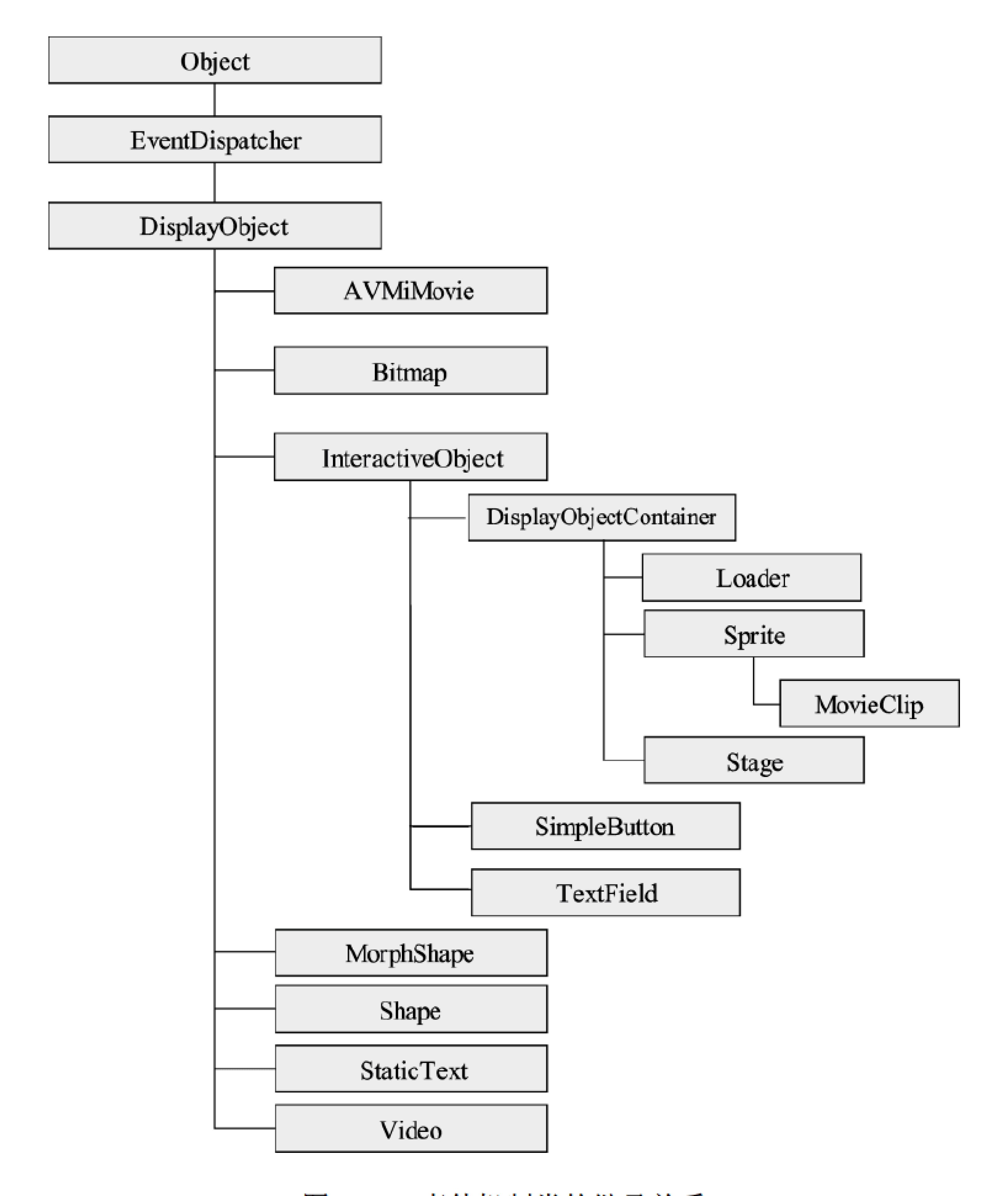


图 12-4　事件机制类的继承关系

表 12-2　事件机制类

事件机制类	说明
Object	Object 类位于 ActionScript 类层次结构的根处。Object 由构造函数使用 new 运算符语法创建，并且可以具有动态赋予属性
EventDispatcher	EventDispatcher 和 IEventDispatcher 是 Flash Player 内置的功能对象，负责实现事件模型，它们同属于 flash.events 包。在这个包中还包括了 Flash Player 的所有系统事件，比如鼠标事件。IEventDispathcer 是一个接口类型的对象，里面定义了一个事件模型中应该具备的基本方法：派发事件和注册监听器。EventDispatcher 则是 IeventDispatcher 上的一具体实现，是 ActionScript 3.0 事件机制的重中之重。EventDispatcher 对象提供了 3 个关键的函数来实现动作事件机制：addEventListener()、removeEventListener()和 dispatchEvent()分别用于注册、移除事件监听器和派发事件
DisplayObject	该类作为一切可视化元素的父类
AVMlMovie	ActionScript 3.0 之前的 SWF 文件或 MC，使用 AVMI 来解释执行。该对象主要用来加载早期的 SWF
Bimap	位图对象，可以通过代码创建，也可以从外部载入

续表

事件机制类	说明
InteractiveObject	可以接受用户交互的对象，抽象类，无法通过代码创建
MorphShape	在 Flash 创建形变动画时，Flash Player 自动生成，不可以通过代码生成
Shape	形状元件，可以使用 ActionScript 创建，通过绘图函数来绘制图形
StaticText	在 Flash 中使用静态文本时会自动创建，不可通过代码创建
Video	视频对象，专门用来播放来自文件或网络的视频，可以由程序创建
DisplayObjectContainer	抽象的可视化对象容器，和容器类控件相似，可以添加其他可视化对象
Loader	加载所有的外部数据
Sprite	场景，位于显示层的顶部，包括程序中所有的可视化元素
MovieClip	扩展了 Sprite，依然和早期的 MC 一样，功能最多最庞杂
Stage	ActionScript 3.0 可视化元素中最重要的一个，专门用来处理代码
SimpleButton	处理按钮的对象
TextField	处理动态文本

12.3 事件机制的工作流程

事件机制的工作流程是本章的重点部分，本节将逐步对其进行介绍。熟练掌握事件机制的工作流程将为理解或创建事件发挥重要作用。

12.3.1 事件流

ActionScript 3.0 的事件模型中新增加了事件流的概念。当一个事件发生时，必然存在一个派发事件的对象，这里称之为目标对象。当事件发生后，Flash Player 生成一个携带数据的对象，然后检查目标对象是否处在显示层中，如果是，则遍历从根容器一直到目标对象所在位置的所有对象。由于 ActionScript 3.0 的显示层是基于容器模式的，这使得显示层类似于一棵树，每个对象都是树上的一个节点。Flash Player 自动检测所经过的节点中是否注册了监听器。

事件流运行流程分为以下 3 步。

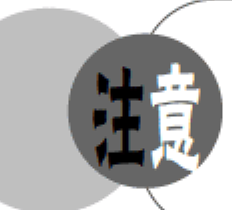
- **捕获阶段**　捕获事件 capturing，从根节点开始顺序而下，检测每个节点是否注册了监听器。同时，Flex 将事件对象的 currentTarget 值改为当前正在检测的对象。如果注册了监听器，则调用监听函数。
- **目标阶段**　检测目标的监听器 targeting，触发在目标对象本身注册的监听程序。
- **冒泡阶段**　事件冒泡 bubbling，从目标节点到根节点，检测每个节点是否注册了监听器，如果有，则调用监听函数。

每个事件对象都有以下属性。

- **Target**　事件的派发者。
- **currentTarget**　当前正在检测的对象，帮助跟踪事件传播的过程。

事件只在 bubbles 属性为 true 时才进行冒泡，可以冒泡的事件包括 change、click、

doubleClick、keyDown、keyUp、mouseDown、mouseUp。并且不能在一个监听器中同时打开捕获和冒泡功能，要做到这一点，只能注册两个监听器，分别实现。



事件发生后，每个节点可以有两个机会（2 选 1）响应事件，默认情况下，捕获功能处于关闭状态，一般没有必要进行捕获跟踪。

通过上面的讲解，大家应该对事件流有了简单的认识，但只有理论知识的讲解显然不容易理解。下面通过一个实例来具体说明事件流的工作流程，步骤如下所示。

（1）打开已经创建的名为 EventDemo 的 Flex 项目，在 src 目录下将添加一个名为 EventDemo1.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 id 为 cav1 的 Canvas 组件，在 cav1 中添加一个 id 为 cav2 的 Canvas 组件，然后再在 cav2 中添加一个 id 为 btn1 的 Button 组件，添加一个 TextArea 组件用于显示提示信息，另外添加一个 id 为 txt 的 TextArea 组件用于输出提示信息。具体如代码 12.4 所示。

代码 12.4　布局窗体：EventDemo1.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete=
"Page_Load()" layout="absolute">
<mx:Canvas x="22" y="55" width="414" height="116" backgroundAlpha="1.0"
backgroundColor="#FB117F" label="画布" id="cav1">
            <mx:Canvas x="43" y="27" width="328" height="69" backgroundColor=
            "#62F50A" id="cav2">
                  <mx:Button x="89" y="19" label="按钮" id="btn1" fontSize=
                  "12" width="150" height="31" fillAlphas="[1.0, 1.0]" fill-
                  Colors="[#ED2121, #ED2121]" color="#FBFDFE"/>
            </mx:Canvas>
      </mx:Canvas>
      <mx:TextArea x="22" y="179" width="664" height="275" id="txt" fontSize=
      "12"/>
      <mx:TextArea x="512" y="55" width="174" height="116" fontSize="24"
      text="1-捕获阶段&#xa;2-目标阶段&#xa;3-冒泡阶段&#xa;"/>
</mx:Application>
```

（3）以上内容添加完毕后，EventDemo1.mxml 窗体布局已完成，各组件间的关系如图 12-5 所示。

（4）分别为按钮 btn1 及画布 cav1、cav2 注册鼠标单击时触发事件，具体如代码 12.5 所示。

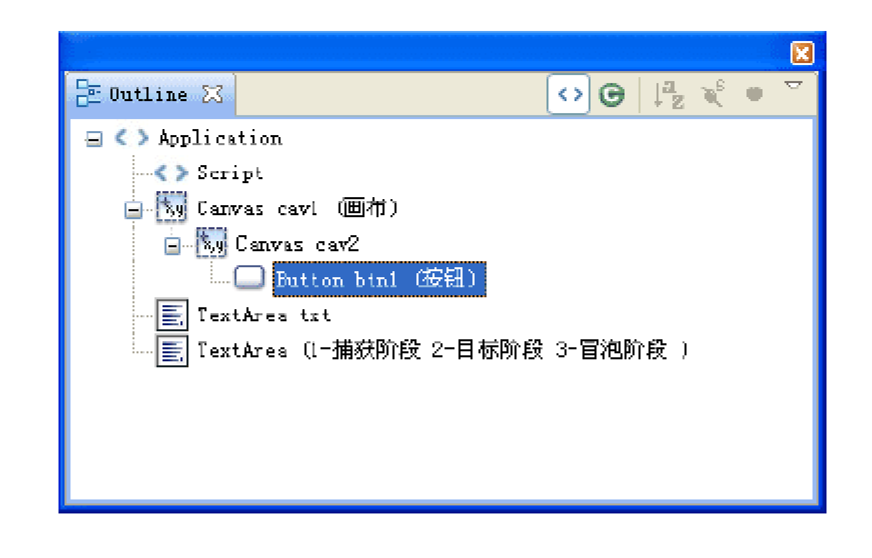


图 12-5　组件间的关系

代码 12.5　注册鼠标事件

```
<mx:Script>
   <![CDATA[
```

```
        internal function Page_Load():void
        {
            cav1.addEventListener(MouseEvent.CLICK,onpress);
            cav2.addEventListener(MouseEvent.CLICK,onpress);
            btn1.addEventListener(MouseEvent.CLICK,onpress);
        }
    ]]>
</mx:Script>
```

（5）当鼠标单击事件触发时检测是否冒泡、目标对象、所处阶段及当前对象等信息，并将信息输出到文本区域中，具体如代码 12.6 所示。

代码 12.6　添加事件触发时执行的操作

```
internal function onpress(evt:MouseEvent):void
{
    ShowMessage("是否冒泡: "+evt.bubbles);
    ShowMessage("目标对象: "+evt.target);
    ShowMessage("所处阶段: "+evt.eventPhase);
    ShowMessage("当前对象: "+evt.currentTarget);
    ShowMessage("---------")
}
internal function ShowMessage(message:String):void
{
    txt.text+=message+"\n";
}
```

监听函数 onpress()中各属性的作用如下所示。

- target 表示派发事件的目标对象。
- currentTarget 表示事件流当前正经过的目标对象。
- bubbles 表示是否打开了冒泡功能。
- eventPhase 表示事件流当前的阶段，捕获、目标或冒泡。

（6）在以上代码添加完成后，关于事件流的工作流程的实例已制作完成。将文件保存后，运行应用程序，页面初始化后的效果如图 12-6 所示。



图 12-6　页面初始化后的效果图

（7）当单击【按钮】按钮时，MouseEvent 默认打开了冒泡功能。事件从上到下属于捕获阶段不触发任何事件，无任何信息输出。然后是 btn2 向上返回的过程，触发在目标对象本身注册的监听程序，调用监听函数，将信息输出到文本区域中，效果如图 12-7 所示。

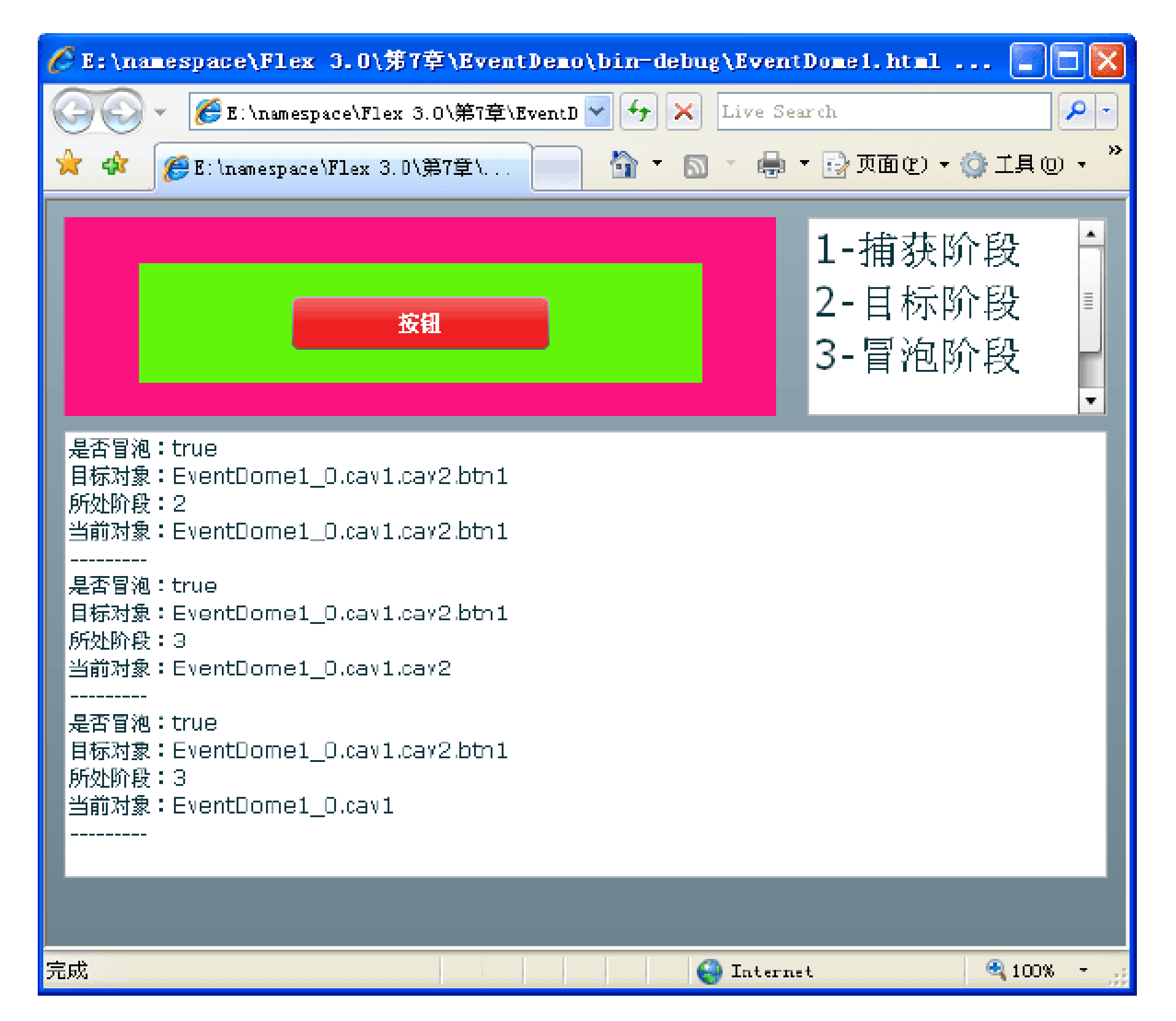


图 12-7　单击【按钮】按钮

（8）页面重新刷新后，效果如图 12-6 所示，单击位于中间位置的 cav2，按照事件流的运行流程，将先后输出 cav2 和 cav1 的信息，效果如图 12-8 所示。同理，如果单击 cav1 则仅输出 cav1 的信息。

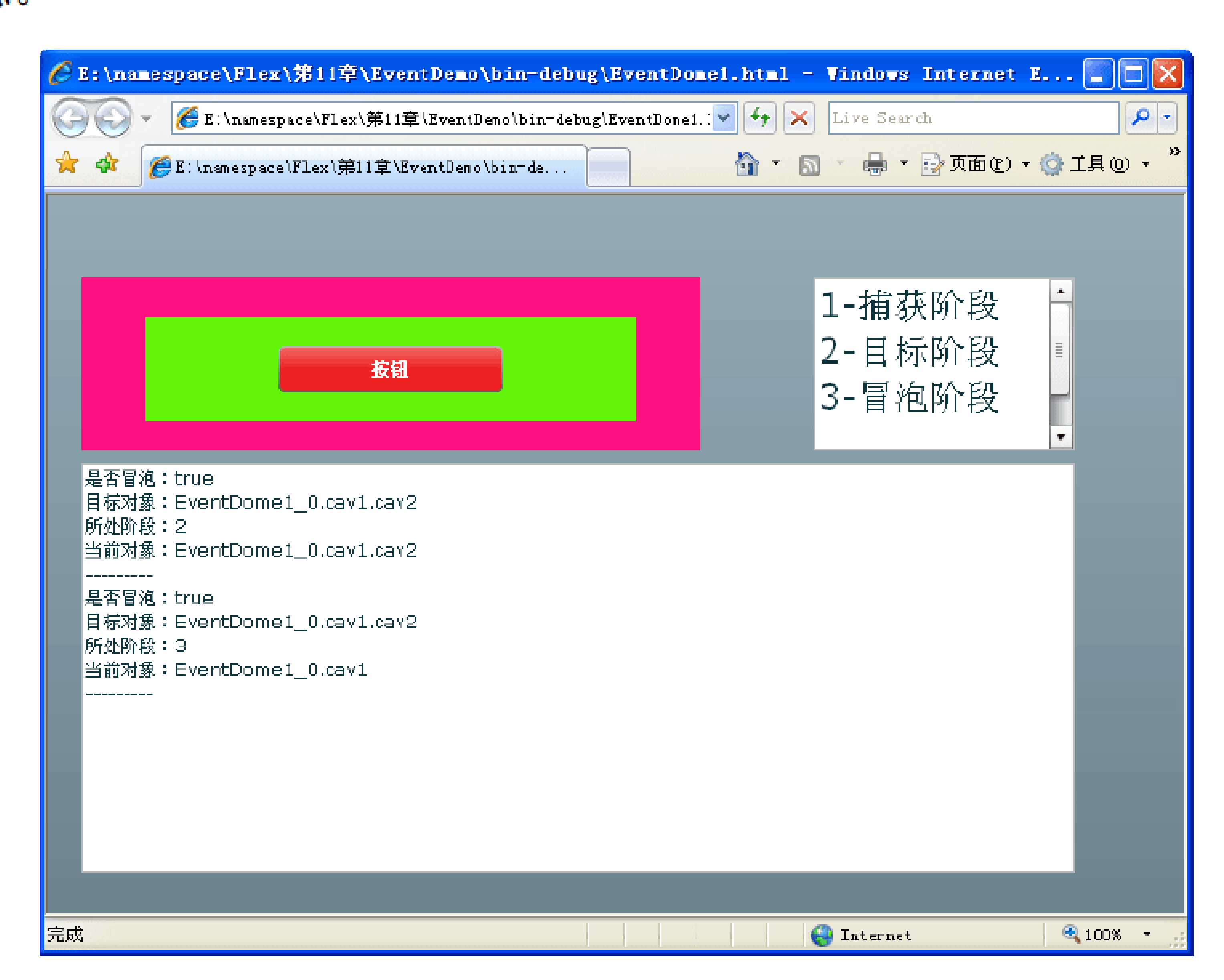


图 12-8　单击位于中间位置的 cav2

（9）在 12.1 节介绍 addEventListener()方法时已经知道，如果 useCapture 设置为 true，打

开了捕获功能，则该组件的冒泡阶段被取消。如果将监听方法的参数设置为捕获功能，则具体方法如代码 12.7 所示。

代码 12.7　将监听方法的参数设置为捕获功能

```
internal function Page_Load():void
{
    cav1.addEventListener(MouseEvent.CLICK,onpress,true);
    cav2.addEventListener(MouseEvent.CLICK,onpress,true);
    btn1.addEventListener(MouseEvent.CLICK,onpress,true);
}
```

（10）运行应用程序，页面加载后，单击【按钮】按钮时，该按钮的冒泡功能被取消。在捕获功能下，按钮不触发任何事件，而流经的两个 Canvas 将触发监听函数 onpress()，输出相关信息，效果如图 12-9 所示。

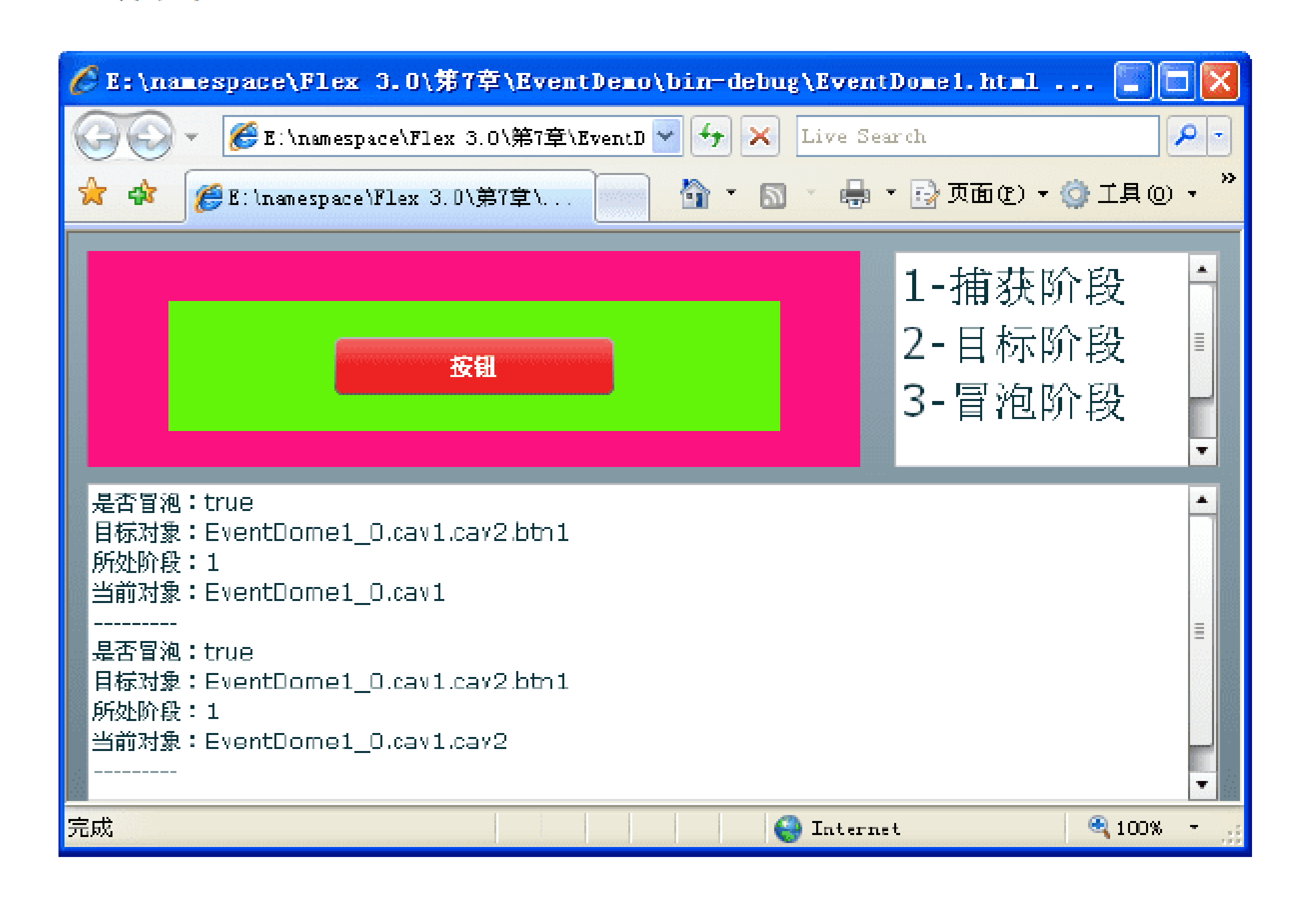


图 12-9　打开捕获功能时的效果

12.3.2　Event 对象概述

Event 类作为创建 Event 对象的基类，当发生事件时，Event 对象将作为参数传递给事件监听器。

Event 类的属性包含有关事件的基本信息，例如事件的类型或者是否可以取消事件的默认行为。对于许多事件（如由 Event 类常量表示的事件），此基本信息就足够了。但其他事件可能需要更详细的信息。例如，与鼠标单击关联的事件需要包括有关单击事件的位置以及在单击事件期间是否按了任何键的其他信息。这可以通过扩展 Event 类（MouseEvent 类执行的操作）将此类其他信息传递给事件监听器。

Event 类的方法可以在事件监听器函数中使用以影响事件对象的行为。某些事件有关联的默认行为。例如，doubleClick 事件有关联的默认行为，此行为突出显示事件发生时鼠标指针下的词。通过调用 preventDefault()方法，事件监听器可以取消此行为。通过调用 stopPropogation()或 stopImmediatePropogation()方法，还可以使当前事件监听器成为要处理事件的最后一个事件

监听器。Event 类的属性和方法如表 12-3、表 12-4 所示。

表 12-3　Event 类属性

属性名称	类型	作用
bubbles	Boolean	[read-only]指示事件是否为冒泡事件
cancelable	Boolean	[read-only]指示是否可以阻止与事件相关联的行为
constructor	Object	对类对象或给定对象实例的构造函数的引用
currentTarget	Object	[read-only]当前正在使用某个事件监听器处理 Event 对象的对象
eventPhase	uint	[read-only]事件流中的当前阶段
prototype	Object	[static]对类或函数对象的原型对象的引用
target	Object	[read-only]事件目标
type	String	[read-only]事件的类型

表 12-4　Event 类方法

方法名称	作用
Event(type:String, bubbles:Boolean = false, cancelable:Boolean = false)	创建一个作为参数传递给事件监听器的 Event 对象
clone():Event	复制 Event 子类的实例
formatToString(className:String,... arguments):String	在自定义 Event 类中实现 toString()方法的实用程序函数
hasOwnProperty(name:String):Boolean	指示对象是否已经定义了指定的属性
isDefaultPrevented():Boolean	检查是否已对事件调用 preventDefault()方法
isPrototypeOf(theClass:Object):Boolean	指示 Object 类的实例是否在指定为参数的对象的原型链中
preventDefault():void	如果可以取消事件的默认行为，则取消该行为
propertyIsEnumerable(name:String):Boolean	指示指定的属性是否存在、是否可枚举
setPropertyIsEnumerable(name:String, isEnum:Boolean = true):void	设置循环操作动态属性的可用性
stopImmediatePropagation():void	防止对事件流中当前节点和所有后续节点中的事件监听器进行处理
stopPropagation():void	防止对事件流中当前节点的后续节点中的所有事件监听器进行处理
toString():String	返回一个字符串，其中包含 Event 对象的所有属性

12.3.3　创建自定义事件

在对自定义事件介绍之前，首先要对 EventDispatcher 类的 dispatchEvent()方法有一定的了解，dispatchEvent()方法格式如下所示。

```
public function dispatchEvent(event:Event):Boolean
```

dispatchEvent()方法将事件调度到事件流中。事件目标是对其调用 dispatchEvent()方法的 EventDispatcher 对象。

参数 event:Event 用于调度到事件流中的 Event 对象。如果正在重新调度事件，则会自动创建此事件的一个克隆。在调度了事件后，其 target 属性将无法更改，因此必须创建此事件的一个新副本以能够重新调度。如果成功调度了事件，则值为 true。值 false 表示失败或对事件调用了 preventDefault()方法。

下面通过一个实例来具体说明如何创建自定义事件，具体步骤如下所示。

（1）打开已经创建的名为 EventDemo 的 Flex 项目，在 src 目录下将添加一个名为 EventDemo2.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。首先添加一个 id 为 cav 的 Canvas 组件，在 cav 的外部添加一个 id 为 btn 的 button 组件。具体如代码 12.8 所示。

代码 12.8　布局窗体：EventDemo2.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete=
"Page_load()" layout="absolute" backgroundGradientAlphas="[1.0, 1.0]"
backgroundGradientColors="[#F60C0C, #FBFBFB]">

    <mx:Canvas x="317" y="22" width="380" height="340" borderColor="#8317AB"
        backgroundColor="#05F81C" id="cav"  backgroundAlpha="1.0"
        cornerRadius="20" label="画布">
    </mx:Canvas>
    <mx:Button x="10" y="130" label="按钮" id="btn"  width="260" height="124"
    fontSize="24" fontStyle="normal" fontWeight="bold"/>
</mx:Application>
```

（3）以上内容添加完毕后，EventDemo2.mxml 窗体布局已完成，各组件间的关系如图 12-10 所示。

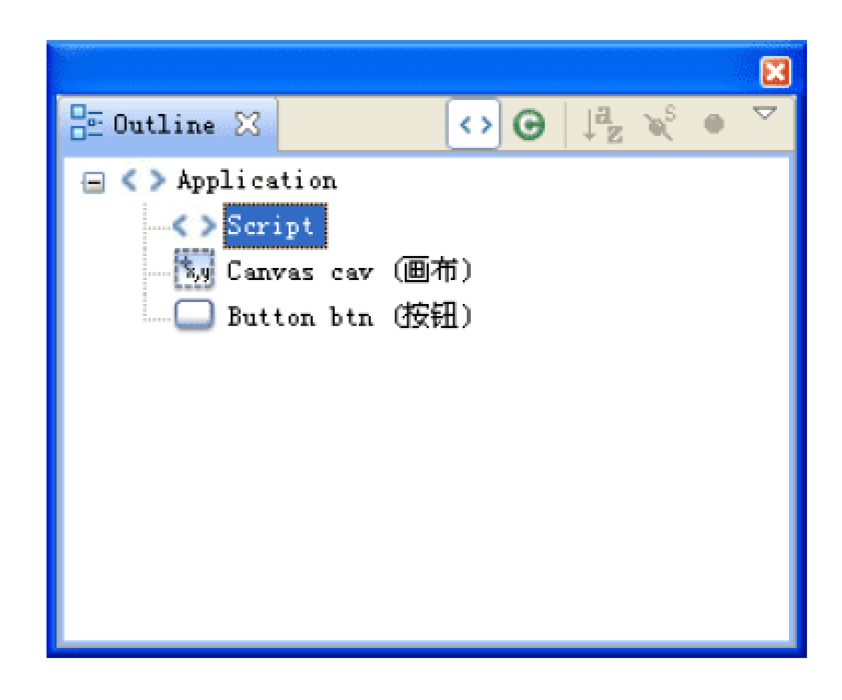


图 12-10　组件间的关系

（4）分别为按钮 btn 和画布 cav 注册 onShow 和 onClick 事件，具体如代码 12.9 所示。

代码 12.9　注册事件

```
<mx:Script>
        <![CDATA[
```

```
                import mx.controls.Alert;

                internal function Page_load():void{
                       cav.addEventListener("testEvent",onShow);
                       btn.addEventListener(MouseEvent.CLICK,onClick);
                       trace("1:addEventlistener");
                }

                internal function onClick(evt:MouseEvent):void{

                              trace("2:dispatchEvent!");
                              cav.dispatchEvent(new Event("testEvent",true,
                              false));
                }

        ]]>
</mx:Script>
```

（5）当 onShow 事件触发时，将输出触发事件的目标对象以及事件的类型，具体如代码 12.10 所示。

代码 12.10　添加事件触发时执行的操作

```
public function onShow(evt:Event):void{
    trace("3:onShow "+evt.type);
    Alert.show("触发了"+evt.currentTarget+"注册的: "+evt.type+"事件");
}
```

（6）在以上代码添加完成后，关于创建自定义事件的实例已制作完成。将文件保存后，单击 Debug 按钮右侧列表中的文件名，启动调试模式。运行应用程序，页面初始化后的效果如图 12-11 所示。

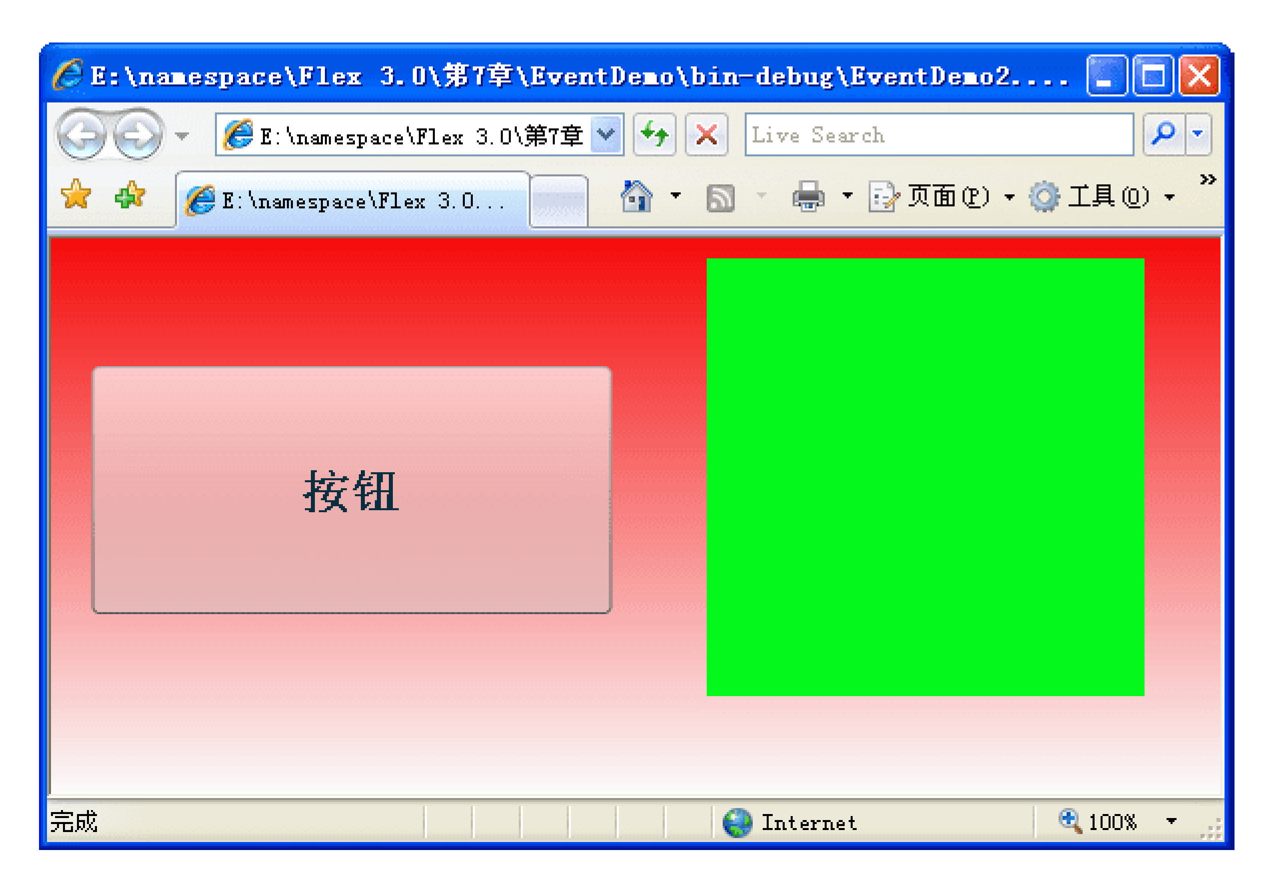


图 12-11　页面初始化效果

（7）在页面加载过程中，Page_Load()方法被调用，在 onShow 及 onClick 事件注册完毕后，输出信息“1：addEventlistener”，该信息输出到 Console 窗格中，效果如图 12-12 所示。

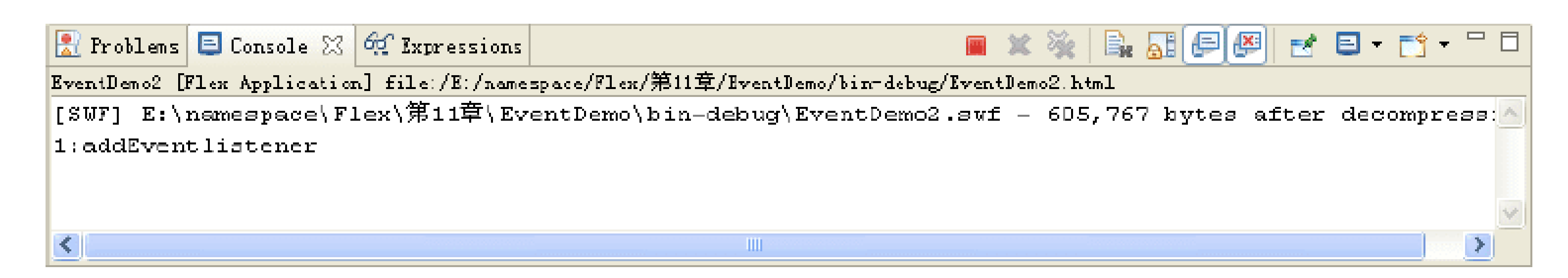


图 12-12　Console 窗格

（8）单击【按钮】按钮，弹出提示信息为“触发了 EventDemo2_0.cav 注册的：testEvent 事件”的提示对话框，效果如图 12-13 所示。此操作说明，在单击按钮之后，派发了一个类型为 testEvent 的事件，该事件成功地被画布所响应并调用了 onShow()方法。

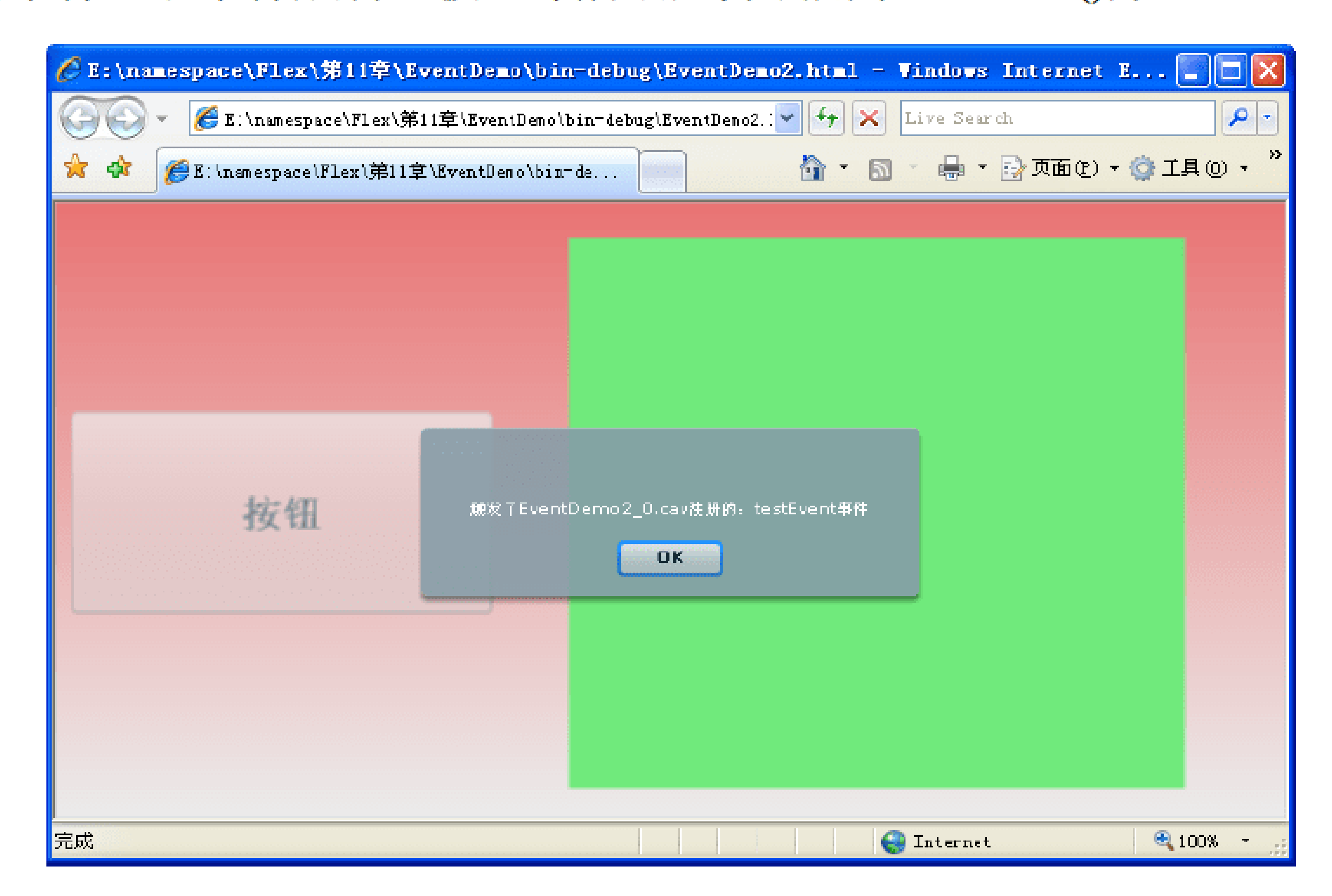


图 12-13　弹出提示对话框

（9）在 onClick()与 onShow()方法被调用的同时，将信息“2:dispatchEvent!”和“3:onShow testEvent”输出到 Console 窗格中，效果如图 12-14 所示。

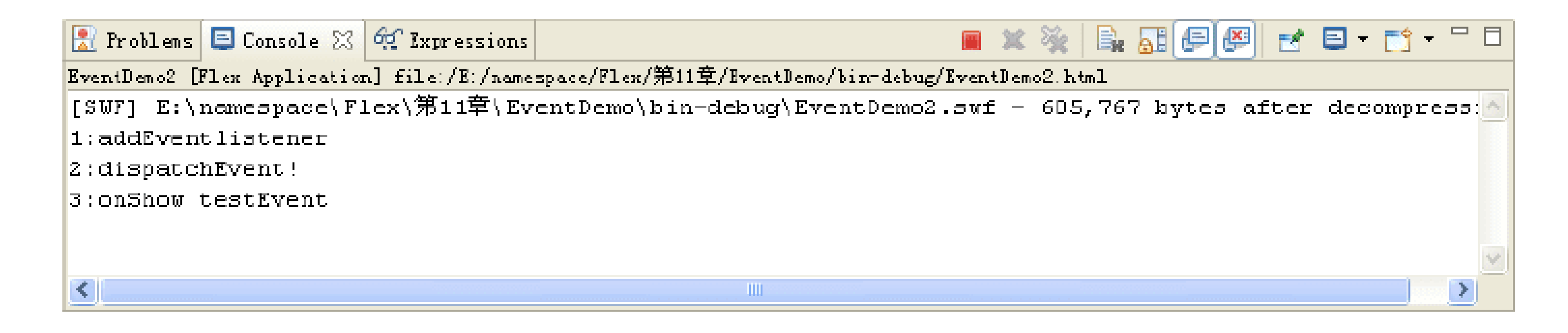


图 12-14　Console 窗格

（10）在定义的 onClick()方法中，使用了 cav.dispatchEvent(new Event("testEvent",true,false))来派发事件，从结果得知派发成功。如果使用应用程序来派发是否会成功呢，下面将 onClick()方法中的代码更改，具体内容为如代码 12.11 所示。

代码 12.11　更改后的 onClick()方法

```
internal function onClick(evt:MouseEvent):void{
```

```
                trace("2:dispatchEvent!");
                this.dispatchEvent(new Event("testEvent",true,false));
                trace(this.toString());
    }
```


（11）使用调试模式重新启动应用程序，页面加载后仍然输出“1:addEventlistener”。再单击【按钮】按钮，页面无任何变化，Console 窗格中输出“2:dispatchEvent!”和“EventDemo2_0”，效果如图 12-15 所示。

```
Problems  Console  Expressions
EventDemo2 [Flex Application] file:/E:/namespace/Flex/第11章/EventDemo/bin-debug/EventDemo2.html
[SWF] E:\namespace\Flex\第11章\EventDemo\bin-debug\EventDemo2.swf - 605,779 bytes after decompress:
1:addEventlistener
2:dispatchEvent!
EventDemo2_0
```

图 12-15　Console 窗格

（12）结果表明，使用应用程序派发类型为 testEvent 的事件未成功，尝试使用按钮 btn 派发也一样。

（13）更改窗体布局，将按钮放入布面中，更改后的窗体布局如代码 12.12 所示。

代码 12.12　更改后的窗体布局

```
<mx:Canvas x="90" y="22" width="607" height="306" borderColor="#8317AB"
       backgroundColor="#05F81C" id="cav"  backgroundAlpha="1.0"
       cornerRadius="20" label="画布">
       <mx:Button x="173.5" y="88" label="按钮" id="btn"  width="260"height=
       "124" fontSize="24" fontStyle="normal" fontWeight="bold"/>
</mx:Canvas>
```

（14）在窗体布局更改完毕后，各组件间的关系如图 12-16 所示。

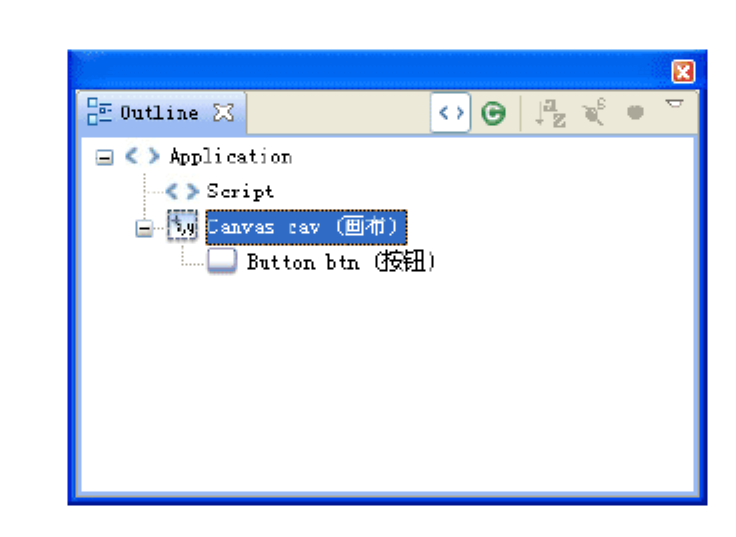


图 12-16　组件间的关系

（15）其他均不变，仍然使用画布 cav 派发了一个类型为 testEvent 的事件，启动调试模式，运行应用程序，页面初始化后单击【按钮】按钮，也弹出提示信息为“触发了 EventDemo2_0.cav 注册的：testEvent 事件”的提示对话框，效果如图 12-13 所示。其结果与布局更改前完全一致。如果再使用按钮 btn 派发了类型为 testEvent 的事件，结果会如何呢？下面将代码更改为按钮 btn 派发，代码如下所示。

```
btn.dispatchEvent(new Event("testEvent",true,false));
```

（16）再次启动调试模式，运行应用程序，页面初始化后效果如图 12-17 所示。

（17）单击【按钮】按钮，弹出提示信息为“触发了 EventDemo2_0.cav 注册的：testEvent 事件”的提示对话框，效果如图 12-18 所示。

图 12-17　页面初始化后效果

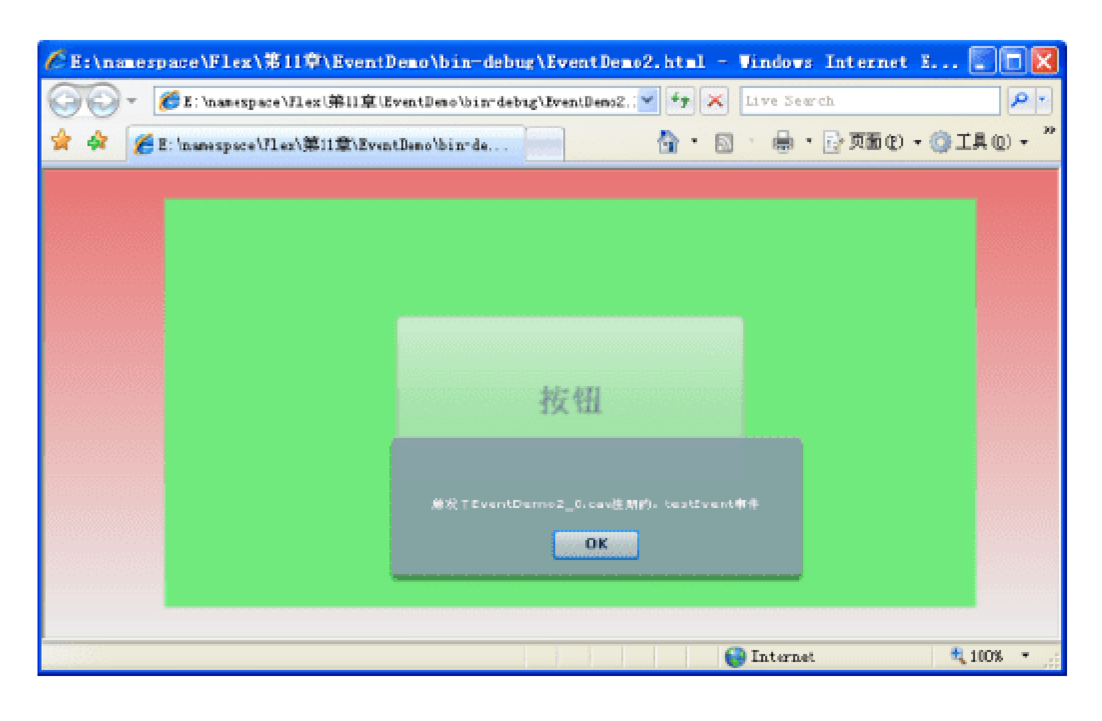


图 12-18　使用按钮 btn 派发

此操作说明，使用按钮 btn 派发了类型为 testEvent 的事件，该事件成功地被画布所响应并调用了 onShow()方法。在前面的讲解中，当使用 btn 派发时并未成功，那为什么在这里使用 btn 派发就成功了呢？这就涉及到上一节中所讲解的事件流，由于按钮在画布的内部，当单击按钮时，产生新的事件对象，该事件对象就会顺着事件流的方向流动。由于默认为冒泡，即从按钮 btn 流到画布 cav 上，而 cav 注册了 onShow()事件，所以做出响应。

（18）为按钮 btn 也添加 addEventListener("testEvent",onShow)监听，Page_Load()方法更改后内容如代码 12.13 所示。

代码 12.13　更改后的 Page_Load()方法

```
internal function Page_load():void{
        cav.addEventListener("testEvent",onShow);
        btn.addEventListener("testEvent",onShow);
        btn.addEventListener(MouseEvent.CLICK,onClick);
        trace("1:addEventlistener");
}
```

（19）再次启动调试模式，运行应用程序，页面初始化后单击【按钮】按钮，弹出了提示信息为“触发了 EventDemo2_0.cav.btn 注册的：testEvent 事件”和“触发了 EventDemo2_0.cav 注册的：testEvent 事件”的两个提示对话框，效果如图 12-19 所示。

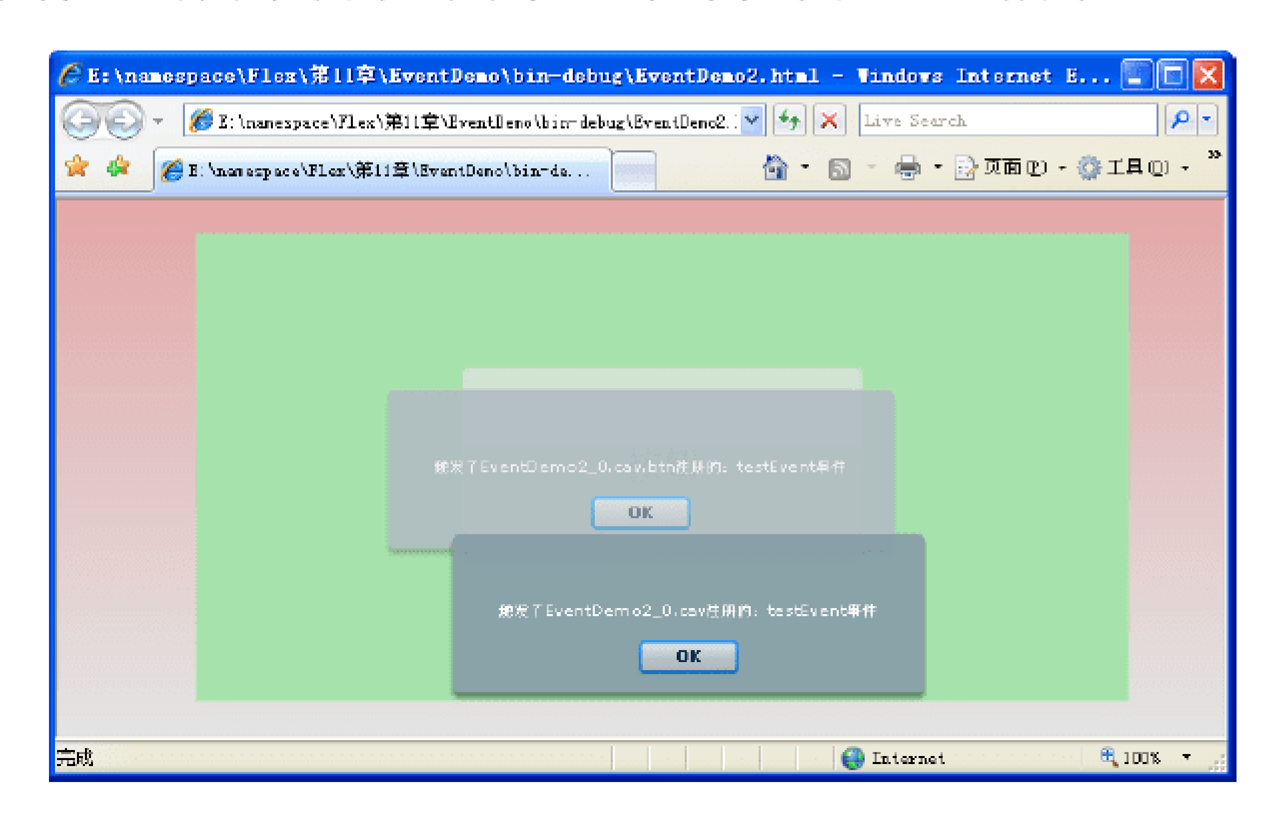


图 12-19　弹出两个提示对话框

这个结果也是由于事件流的缘故，上面已经提及到，由于默认为冒泡，即从按钮 btn 流到画布 cav 上。因为按钮已经注册了 onShow 事件，所以先被执行弹出对话框，而后再执行 cav 注册的 onShow 事件，弹出第二个对话框。

（20）如果想让单击按钮时不影响画布注册的 onShow 事件，该如何去做呢？下面在 onShow()方法中加入判断语句，该方法更改后如代码 12.14 所示。

代码 12.14　onShow()方法更改的代码：onShow()

```
public function onShow(evt:Event):void{
    trace("3:onShow "+evt.type);
    if(evt.currentTarget==evt.target){
        Alert.show("触发了"+evt.currentTarget+"注册的: "+evt.type+"事件");
    }
}
```

（21）再次启动调试模式，运行应用程序，页面初始化后单击【按钮】按钮，弹出了提示信息为“触发了 EventDemo2_0.cav.btn 注册的：testEvent 事件”的提示对话框，效果如图 12-20 所示。

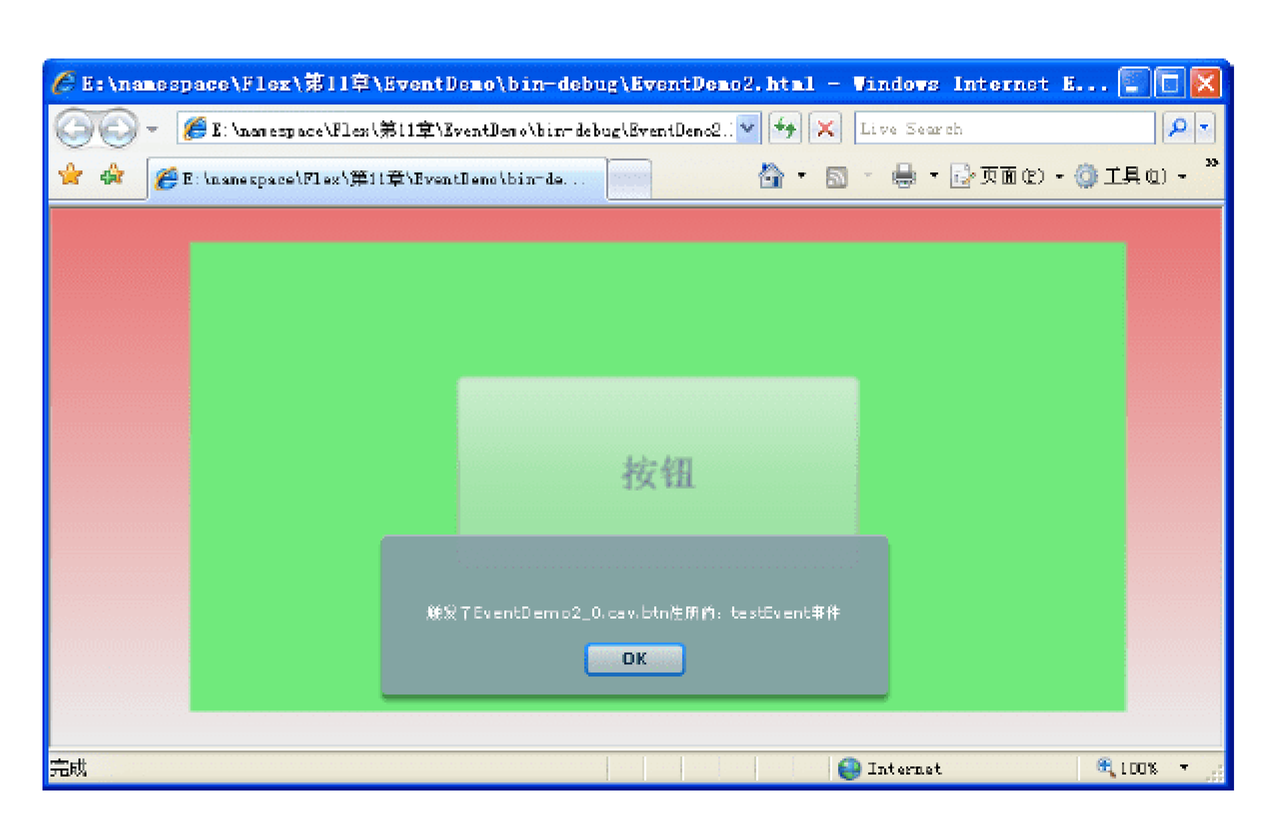


图 12-20　弹出提示对话框

判断语句中的 currentTarget 表示事件触发的目标对象，而 Target 表示事件流流经的对象，因此可以确保单击按钮时只触发目标对象按钮所注册的事件。

12.3.4　扩展自定义事件

上一节中已经详细介绍了如何创建自定义事件，本节将介绍如何在事件中进行参数的传递。Event 是一个对象，包含了一些必要的信息供监听函数使用。开发者如何给自定义的事件中加入参数呢？解决方法很简单，在子类中加入添加参数的接口，派发事件时，直接发送实例，这样就实现了数据的添加。

下面通过一个实例来具体说明如何在事件中进行参数的传递，具体步骤如下所示。

（1）打开已经创建的名为 EventDemo 的 Flex 项目，在 src 目录下将添加一名为 EventDemo3.mxml 的 MXML Application 文件。

（2）在该文件中添加 MXML 布局代码。该布局与 EventDemo3.mxml 基本相似，首先添

加一个 id 为 cav 的 Canvas 组件，在 cav 的外部添加一个 id 为 btn 的 button 组件。具体如代码 12.15 所示。

代码 12.15　布局窗体：EventDemo3.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete=
"Page_load()" layout="absolute" backgroundGradientAlphas="[1.0, 1.0]"
backgroundGradientColors="[#0866B6, #FBFBFB]">
    <mx:Canvas x="317" y="22" width="380" height="340" borderColor="#8317AB"
        backgroundColor="#F87D05" id="cav"  backgroundAlpha="1.0"
        cornerRadius="20" label="画布">
    </mx:Canvas>
    <mx:Button x="10" y="130" label="按钮" id="btn"  width="260" height="124"
    fontSize="24" fontStyle="normal" fontWeight="bold" fillAlphas="[1.0, 1.0]"
    fillColors="[#FB0404, #F9F0F0]"/>

</mx:Application>
```

（3）布局完成后，在程序中新建一个名为 DefinedEvent.as 的类文件，将该文件放入 com 文件夹下的 events 文件夹中，具体如代码 12.16 所示。

代码 12.16　创建自定义事件：DefinedEvent.as

```
package com.events
{
    import flash.events.Event;
    //扩展 Event 对象
    public class DefinedEvent extends Event
    {
        //定义静态常量保存事件类型
        public static const DefinedString:String="DefinedString";
        //被传递的参数
        public var ClickedTime:String;
        //构造函数
        public function DefinedEvent(type:String,clickedtime:String){
            //用变量保存参数
            this.ClickedTime=clickedtime;
            //相当于创建了一个 Event 对象，设置 bubbles 和 cancelable 为 false
            super(type, false, false);
        }
    }
}
```

（4）分别为按钮 btn 和画布 cav 注册 onShow 和 onClick 事件，具体如代码 12.17 所示。

代码 12.17　注册事件

```
<mx:Script>
    <![CDATA[
        //导入新创建的自定义事件
        import com.events.DefinedEvent;
        import mx.controls.Alert;
        internal function Page_load():void{
            //注册画布触发事件
            cav.addEventListener(DefinedEvent.DefinedString,onShow);
            //注册按钮单击事件
            btn.addEventListener(MouseEvent.CLICK,onClick);
        }
        //按钮单击事件
        internal function onClick(evt:MouseEvent):void{
            //创建时期对象
            var datetime:Date=new Date();
            //使用时期对象获取当前的日期时间
            var getdate:String=datetime.getFullYear()+"年"+datetime.
            getMonth()+"月"+datetime.getDate()+"日"+datetime.getHours()+"点
            "+datetime.getMinutes()+"分"+datetime.getMinutes()+"秒";
            cav.dispatchEvent(new DefinedEvent(DefinedEvent.DefinedString,
            getdate));
        }
    ]]>
</mx:Script>
```

（5）当 onShow 事件被触发时，将输出触发事件的目标对象以及事件的类型，具体如代码 12.18 所示。

代码 12.18　onShow 事件触发时执行的操作

```
//画布触发事件
internal function onShow(evt:DefinedEvent):void{
    Alert.show("触发了 cav 注册的："+evt.type+"事件,通过事件对象传递的数据是："+evt.
    ClickedTime,"提示");
}
```

（6）在以上代码添加完成后，如何在事件中进行参数传递的实例已制作完成。将文件保存后，运行应用程序，页面初始化后的效果如图 12-21 所示。

（7）单击【按钮】按钮，弹出提示信息为“触发了 cav 注册的：DefinedString 事件，通过事件对象传递的数据是：2008 年 10 月 4 日 13 点 32 分 32 秒”的【提示】对话框，效果如图 12-22 所示。此操作说明，在单击按钮之后，派发了一个类型为 DefinedEvent 的事件，该事件成功的被画布所响应并调用了 onShow()方法。

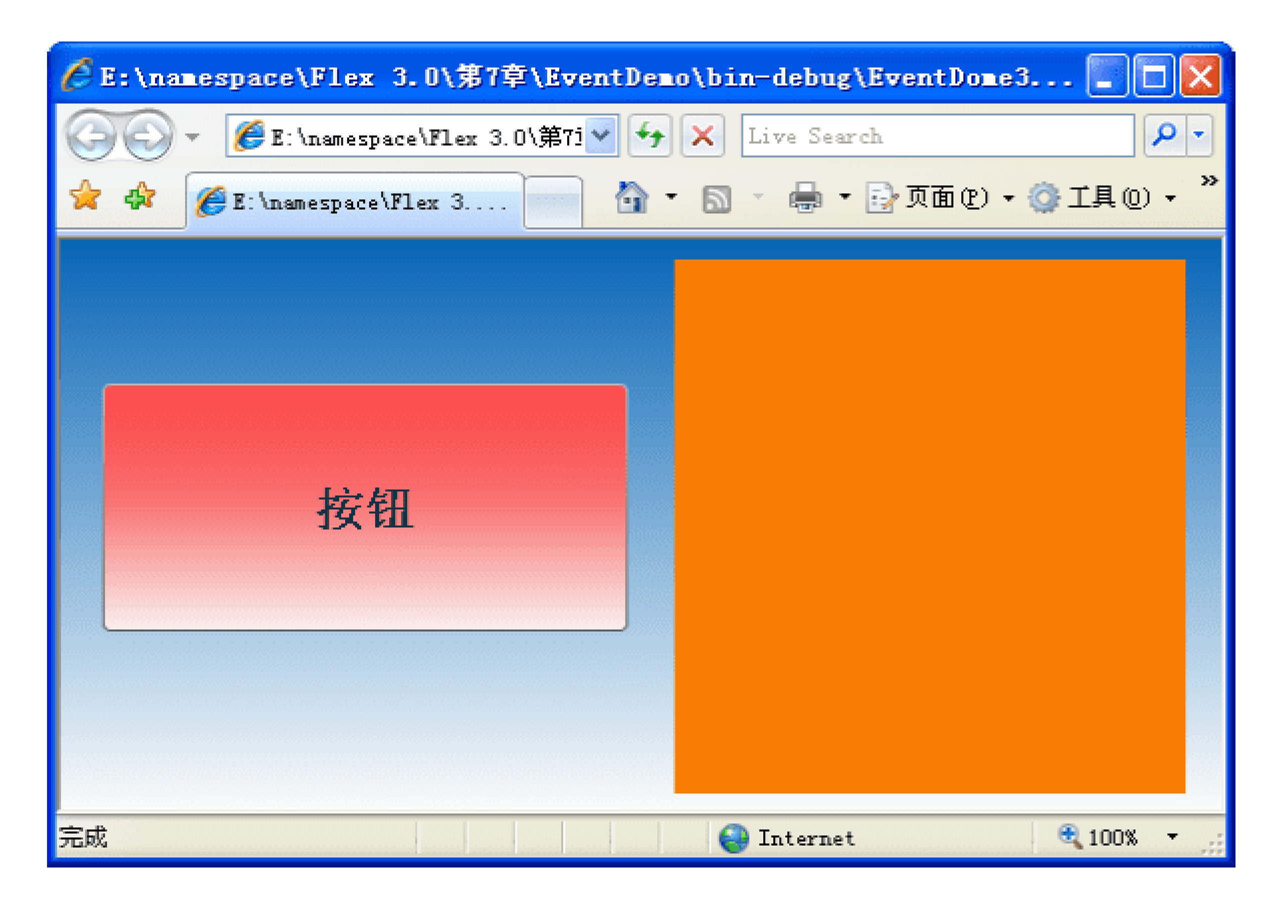


图 12-21　页面初始化效果

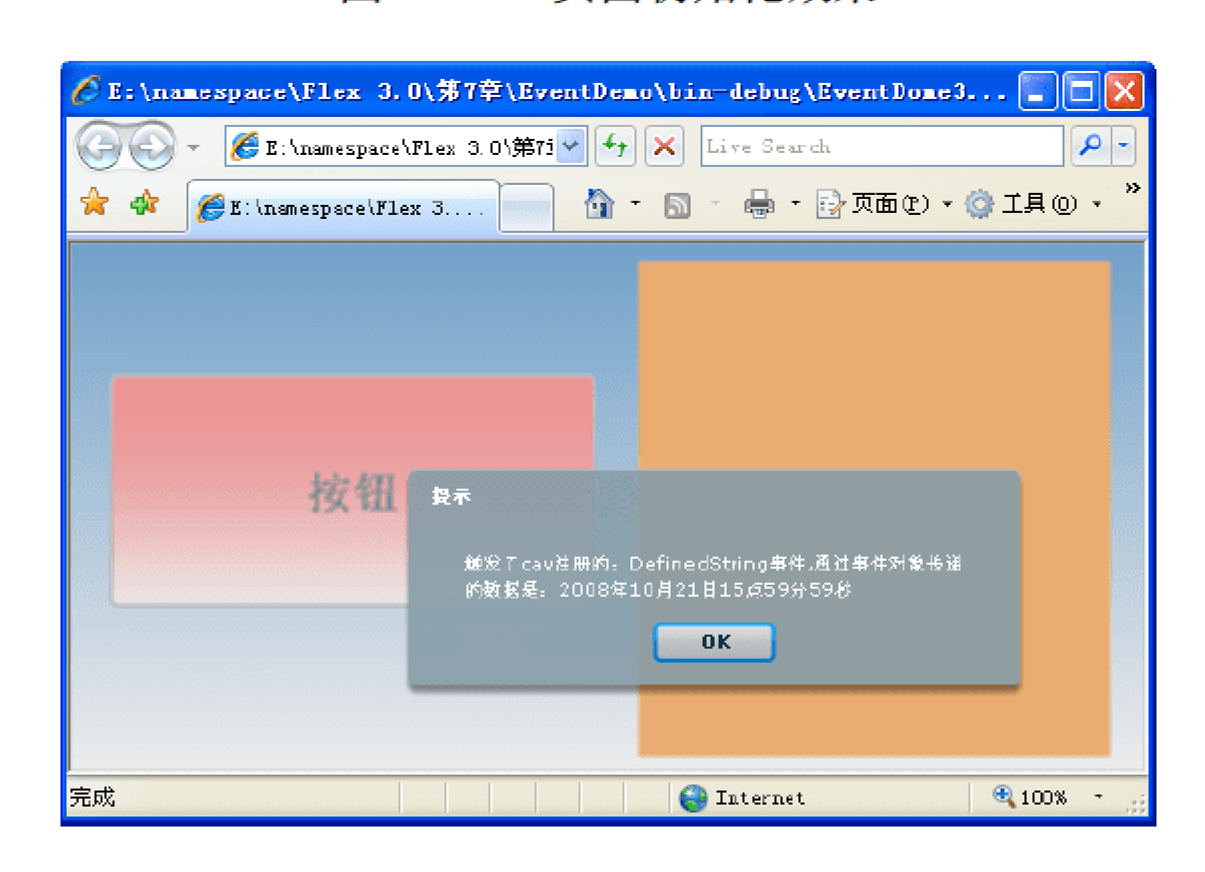


图 12-22　弹出【提示】对话框

12.4　事件机制的高级应用

本节介绍如何使用 MXML 的标签来完成事件处理，与上一节所讲述的内容有一定的区别。前面所讲解的实例中，事件处理完全放在脚本里面进行处理，其实通过配置 MXML 文件的标签和属性也可以实现对事件处理的能力。

下面通过一个用户登录的实例来具体说明如何使用 MXML 的标签来完成事件处理，具体步骤如下所示。

（1）创建一个名为 EventMXMLDemo 的 Flex 项目，在 src 目录下将自动添加一个名为 EventMXMLDemo.mxml 的 MXML Application 文件。

（2）在本实例中，登录窗口以用户自定义组件的方式实现。右击项目名称，从弹出的快捷菜单中选择 New |MXML Component 命令，弹出 New MXML Component 对话框，该组件建立在 Panel 基础上，文件名为 LoginControl，效果如图 12-23 所示。

（3）在 LoginControl.mxml 文件创建完成后，接下来就要布局登录组件了，添加相关的 Lable、TextInput 和 Button 组件，具体如代码 12.19 所示。

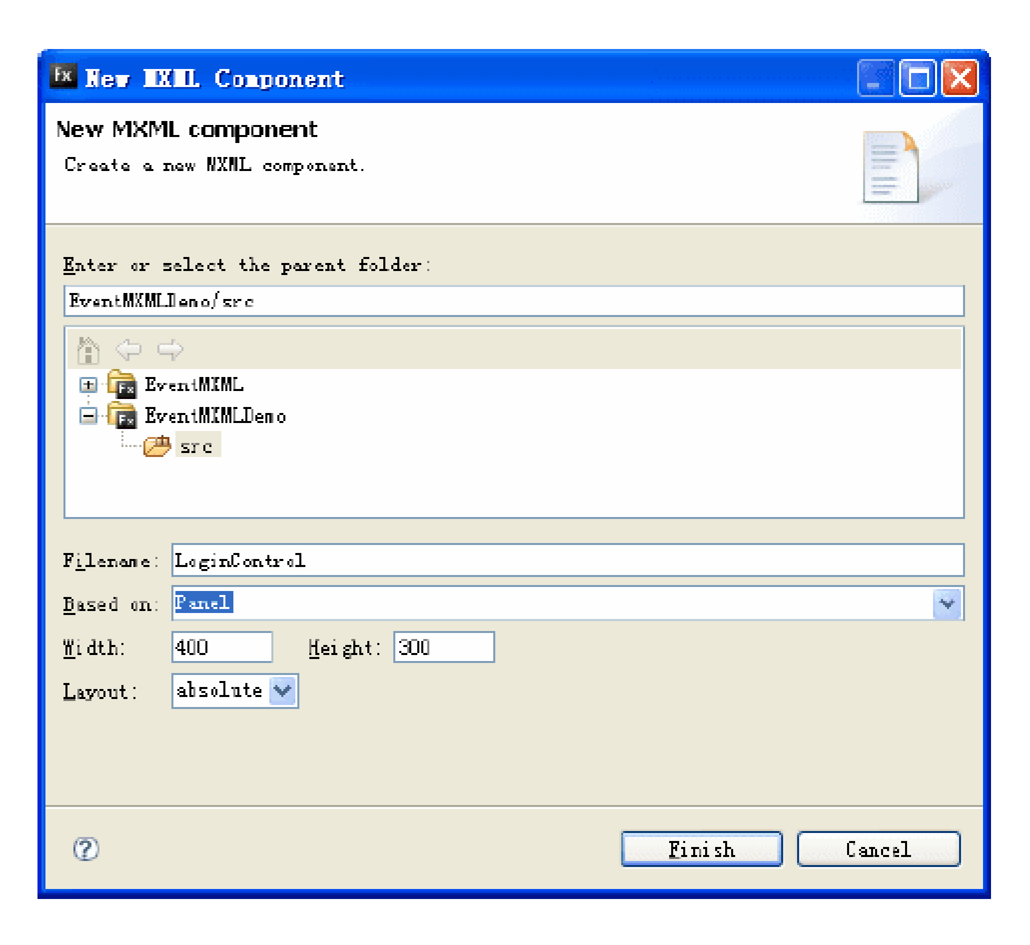


图 12-23　新建 LoginControl 组件

代码 12.19　布局登录组件：LoginControl.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" width=
"450" height="236" cornerRadius="19" borderColor="#7FE25D" fontSize="12"
title="用户登录">
    <mx:Label x="41" y="30" text="用户名: " width="74" height="22"/>
    <mx:Label x="41" y="76" text="密    码: " width="74"/>
    <mx:TextInput x="123" y="28" id="txtUserName" maxChars="15" width="221"
    cornerRadius="20"/>
    <mx:TextInput x="123" y="74" width="221" displayAsPassword="true" id=
    "txtPwd" maxChars="15" cornerRadius="20"/>
    <mx:Button x="34" y="133" label="登    录" width="151" height="33"
    cornerRadius="20" id="btnLogin" click="onClick(event)"/>
    <mx:Button x="223" y="133" label="取    消" width="151" height="33"
    cornerRadius="20" id="btnEsc"/>

</mx:Panel>
```

（4）在组件布局完成后，还需要定义一个登录事件，该事件命名为 LoginEvent，具体如代码 12.20 所示。

代码 12.20　定义 LoginEvent 事件

```
<mx:Metadata>
    [Event(name="LoginEvent",type="flash.events.Event")]
</mx:Metadata>
```

（5）该事件需要由单击按钮时产生的事件流在流经 Panel 时触发，btnLogin 按钮的 onClick 事件如代码 12.21 所示。

代码 12.21　btnLogin 按钮的 onClick 事件

```
<mx:Script>
    <![CDATA[
```

```
        internal function onClick(evt:MouseEvent):void{
            this.dispatchEvent(new Event("LoginEvent"));
        }
    ]]>
</mx:Script>
```

（6）以上内容添加完毕后，登录组件布局已完成，各组件间的关系如图 12-24 所示。

（7）至此，用户登录组件添加完成，最终布局效果如图 12-25 所示。



图 12-24　登录组件中各组件间的关系



图 12-25　登录组件布局效果

（8）在自定义的登录组件制作完成后，可以在 Components 组件窗口中的 Custom 目录下找到。自定义组件的使用也很简单，跟其他自带组件的使用方法完全相同，可以直接拖动，也可以手动输入。在 EventMXMLDemo.mxml 窗体中添加已经定义完成的登录组件，具体如代码 12.22 所示。

代码 12.22　布局窗体：EventMXMLDemo.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="horizon-
tal" xmlns:local="*" backgroundGradientAlphas="[1.0, 1.0]" backgroundGradien-
tColors="[#F30A0A, #FFFFFF]">
    <local:LoginControl id="login1" LoginEvent="Login(event)" layout=
    "absolute">
    </local:LoginControl>
</mx:Application>
```

（9）为 LoginControl 控件定义的 LoginEvent 事件添加触发事件 Login，如果能够成功触发则弹出提示信息为“测试成功！”的【提示】对话框，具体如代码 12.23 所示。

代码 12.23　添加触发事件

```
<mx:Script>
    <![CDATA[
        import mx.controls.Alert;
        internal function Login(evt:Event):void{
            Alert.show("测试成功！","提示")
        }
    ]]>
</mx:Script>
```

（10）在以上代码添加完成后，如何使用 MXML 的标签来完成事件处理的实例已制作完成。将文件保存后，运行应用程序，页面初始化后的效果如图 12-26 所示。



图 12-26　页面初始化效果

（11）单击【登录】按钮，弹出提示信息为“测试成功！”的【提示】对话框，说明 LoginControl 控件定义的 LoginEvent 事件已成功触发 Login 事件。效果如图 12-27 所示。



图 12-27　测试成功

显然用户登录的功能并不完整，这不是用户所需要的，下面对用户登录功能进行完善。

（12）在程序中新建一个名为 LoginFormEvent.as 的 ActionScript 类文件。该类非常简单，仅包括两个全局的 String 类型变量。将该文件放入 com 目录下的 events 目录中，具体如代码 12.24 所示。

代码 12.24　创建自定义事件：LoginFormEvent.as

```
package com.events
{
    import flash.events.Event;

    public class LoginFormEvent extends Event
    {
```

```
            public var username:String="";
            public var password:String="";
            public function LoginFormEvent(type:String)
            {
                super(type, false, false);
            }
        }
    }
```

（13）修改自定义组件中定义的登录事件 LoginEvent，具体如代码 12.25 所示。

代码 12.25　修改定义的 LoginEvent 事件

```
<mx:Metadata>
    [Event(name="LoginEvent",type="com.events.LoginFormEvent")]
</mx:Metadata>
```

（14）为了能够将文本框中的值作为参数传入 LoginFormEvent.as 文件的全局变量中，需要修改自定义组件中 btnLogin 按钮的 onClick 事件，具体如代码 12.26 所示。

代码 12.26　修改 btnLogin 按钮的 onClick 事件

```
<mx:Script>
    <![CDATA[
        import com.events.LoginFormEvent;
        internal function onClick(evt:MouseEvent):void{
            var e:LoginFormEvent=new LoginFormEvent("LoginEvent");
            e.username=txtUserName.text;
            e.password=txtPwd.text;
            this.dispatchEvent(e);
        }
    ]]>
```

（15）在 EventMXMLDemo.mxml 文件中引入已经定义的 com.events.LoginFormEvent 事件。当事件 LoginEvent 被触发时，可以根据传入的参数来判断用户是否合法，若合法则登录成功，否则登录失败。修改后的代码如代码 12.27 所示。

代码 12.27　修改 EventMXMLDemo.mxml 的 Login 事件

```
<mx:Script>
    <![CDATA[
        import mx.controls.Alert;
        import com.events.LoginFormEvent;
        internal function Login(evt:LoginFormEvent):void{
            //Alert.show("测试成功！","提示")
            if(evt.username=="eastjia2008"&&evt.password=="123456"){
```

```
                    Alert.show("用户"+evt.username+"登录成功! ","欢迎光临");
                }
                else{
                    Alert.show("用户"+evt.username+"登录失败! 用户名或密码错误","请重
                    试");
                }
            }
        ]]>
    </mx:Script>
```

（16）在以上代码添加、修改完成后，用户登录的实例已补充完整。将文件保存后，重新运行应用程序，页面加载后，分别在用户名和密码对应的文本框中输入 eastjia2008、123456，效果如图 12-28 所示。



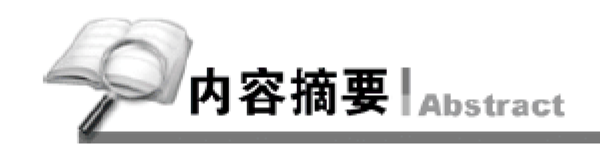
图 12-28　输入用户名和密码

（17）单击【登录】按钮，弹出【欢迎光临】对话框提示“用户 eastjia2008 登录成功！”，效果如图 12-29 所示。



图 12-29　登录成功时弹出【欢迎光临】对话框

第13章 自定义组件



在 Flex 程序开发中，经常会遇到某种类型功能会在程序中多次应用的情况。当遇到这种情况时，如果开发人员不做任何处理，一旦遇到该功能就需要重复编写同样的代码，这样既浪费程序员的时间，也会延迟项目的开发进度，显然是很不明智的做法。要解决此问题并不难，Flex 中为开发人员提供了一种机制，那就是开发人员可以开发自定义组件，将常用功能集合到一起。在程序开发的过程中还会遇到一种情况，那就是组件的功能不能满足项目的要求，此时就要求开发人员开发自定义组件，以满足项目功能的需求。本章将介绍如何在 Flex 中创建自定义组件和使用自定义组件。

学习目标 Objective

- 掌握创建自定义组件的两种方法
- 掌握在组件中添加项目的方法
- 能够编写脚本在程序中动态添加项目
- 掌握 CSS 样式的语法
- 熟悉创建 CSS 样式文件的方法
- 能够编写样式文件和样式
- 能够在程序中使用样式
- 熟悉主题的创建和使用方法
- 掌握组件参数的传递方法

13.1 创建组件

在程序开发中，尤其是一个大型项目中，开发人员不会把所有的代码都塞进一个文件中。基于组件的开发模式是 Flex 的一个特色。一个 Flex 程序是由若干个组件构成的。程序中所有的 MXML 文件和 ActionScript 类文件，都被当作自定义的组件。用户自定义的组件和 Flex 本身的组件在用法上完全一样，它们的区别在于：Flex 组件经过封装，可以被任意程序使用，而用户自定义的组件在特定的程序中才可以使用。

一般将程序中功能独立或者需要重复使用的部分，定义成一个自定义组件。编写程序时，

应当尽量减少组件与组件之间的直接联系，降低块与块之间的依赖性。Flex Builder 3 提供了两种创建自定义组件的方法，包括使用 MXML 和使用 ActionScript。

13.1.1　使用 MXML 创建组件

使用 MXML 创建组件必须在可视化组件的基础上，可以在这些组件中添加标签或其他控件。然后可以将此组件作为一个单元单独对待，为其定义属性和方法。自定义组件可以使程序员很容易地实现代码共享和重复使用，缩短开发周期。自定义组件在第一次调用时被加载到客户端，从而减少了带宽占有率，使程序更加流畅。

在程序开发中，用户登录可能要在许多页面中用到。下面，通过在 Flex Builder 3 中，创建用户登录组件，介绍使用 MXML 创建组件的过程。

首先在 Flex Builder 3 中，选择 File|New|Flex Project 命令，创建一个名称为 chap13 的 Flex 项目。

然后在该项目的 src 目录下创建文件夹，命名为 Components。可以选择 File|New|Folder 命令，也可以在 Flex Navigator 窗格中，右击项目中的 src 目录，选择 New|Folder 命令，打开 New Folder 对话框，效果如图 13-1 所示。

在 Folder name 文本框中输入要创建的文件夹名，单击 Finish 按钮完成创建。也可以在 Windows 资源管理器中，打开项目所在的目录，直接在 src 目录中新建文件夹。创建成功后，在 Navigator 窗格中，显示出创建的文件夹，如图 13-2 所示。

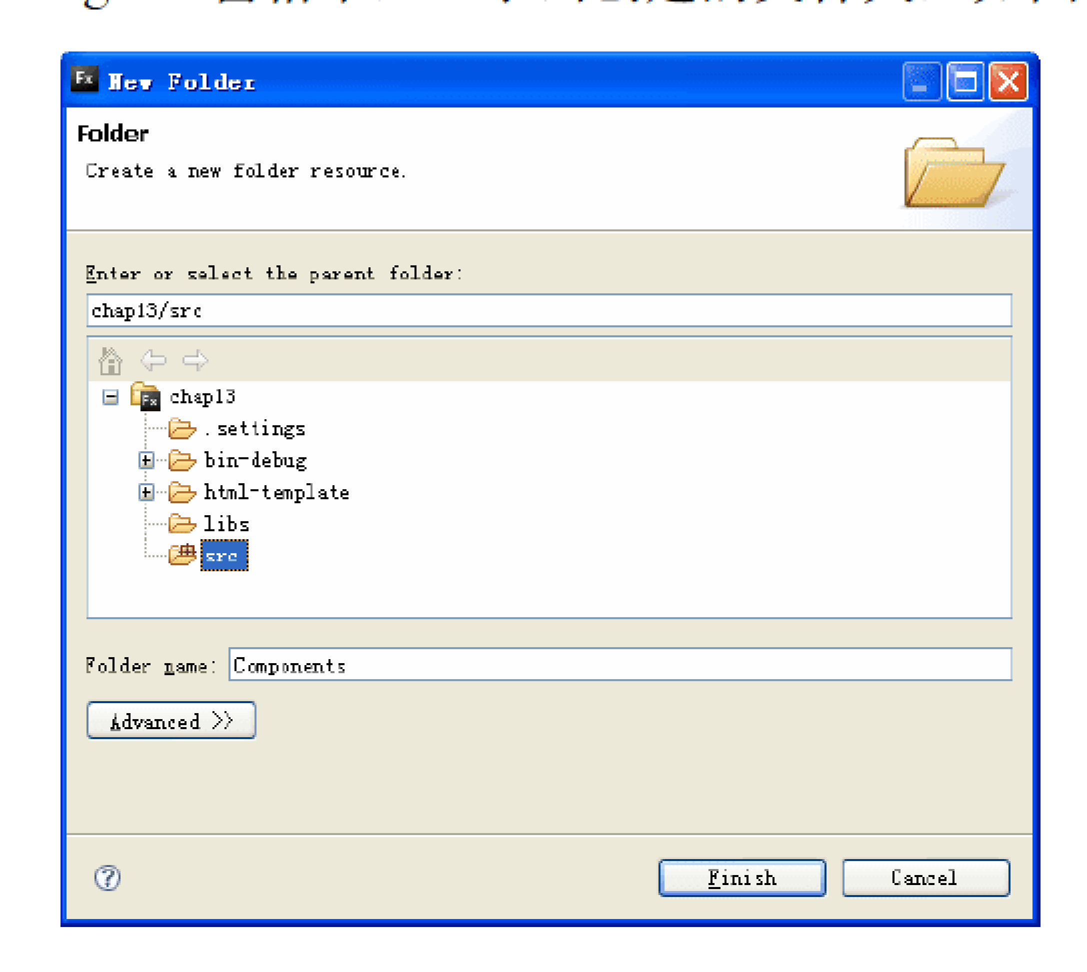



图 13-1　New Folder 对话框

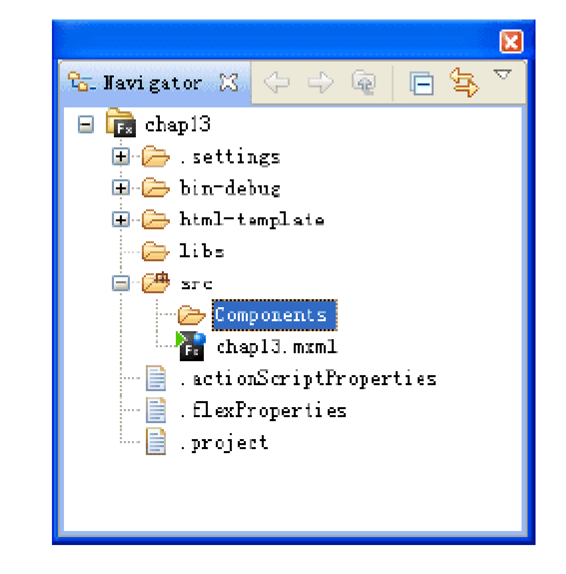


图 13-2　创建 Components 文件夹

创建 Components 文件夹，可以将开发的组件全部存入其中，方便管理。文件夹名也可以根据个人需要创建，但是不能为系统关键字，如 mx 等。

下面在 Components 文件夹中创建组件。在 Navigator 窗格中，右击 Components 文件夹，选择 New|MXML Component 命令，弹出创建 MXML 组件的对话框，如图 13-3 所示。

在 Filename 文本框中，输入要创建组件的名称，例如“myLogin”。从 Based on 下拉列表框中，选择要扩展的组件，这里用到的是 TitleWindow 组件。可以扩展的组件包括 Flex 自带

组件和已经创建的自定义组件，这些组件都是可视化组件。输入长度和宽度，选择布局方式后，单击 Finish 按钮完成创建。

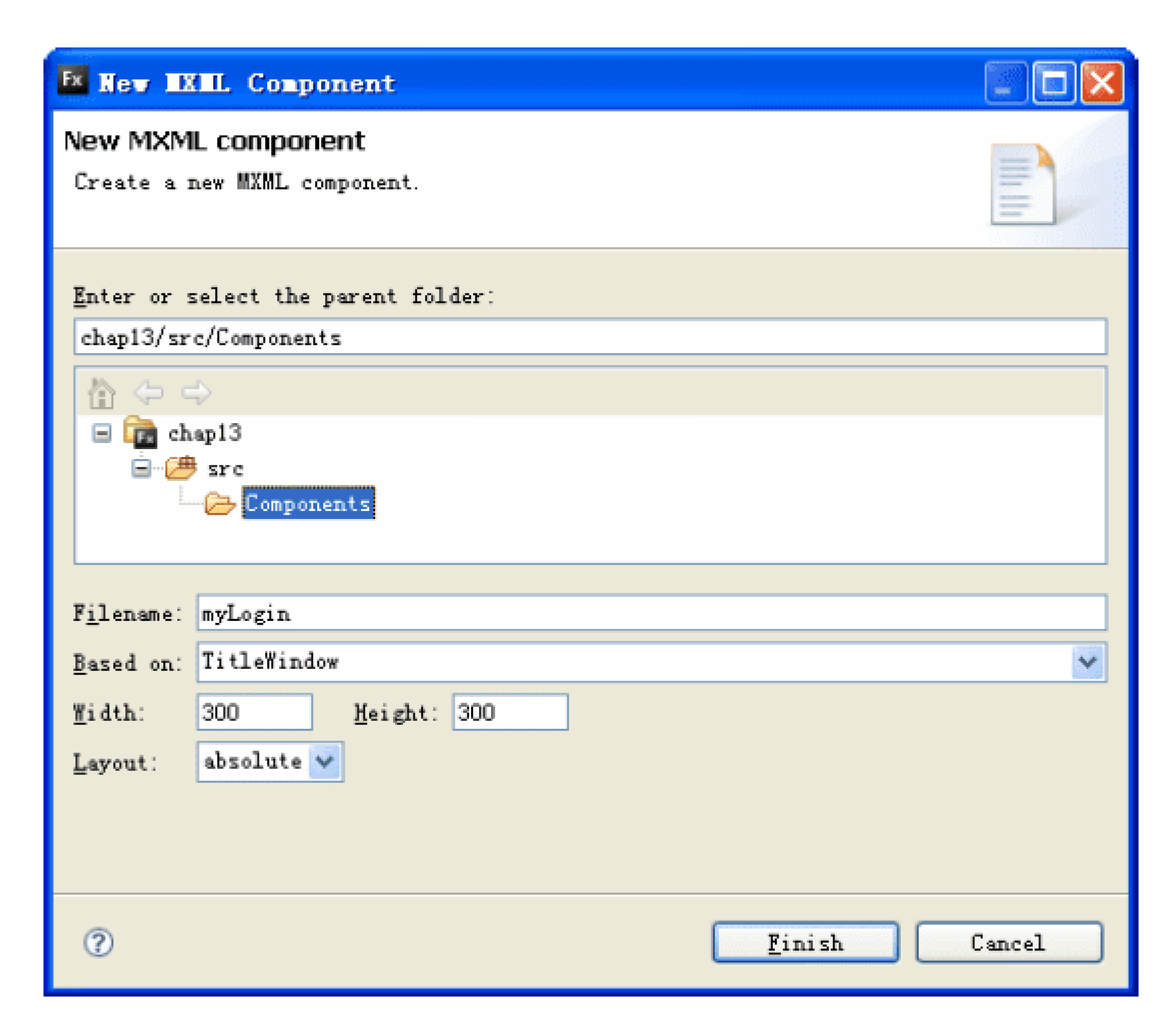


图 13-3　创建 MXML 组件对话框

系统在 Components 文件夹中，自动创建 myLogin.mxml 文件。该文件与 MXML 应用程序文件名后缀一样，但是组件文件不包括<mx:Application>标签，此文件不可单独执行。创建的 myLogin.mxml 文件，内容为一空的 TitleWindow 组件，产生的代码如代码 13.1 所示。

代码 13.1　创建基于 TitleWindow 的组件代码

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="300" height="244">
</mx:TitleWindow>
```

除了可以创建 Component 文件外，还能创建 Module 文件。Module 文件不是基于各种可视化组件，而是基于 Module 的。Module 和 Application 都继承自 LayoutContainer，也就是说 Module 文件本身就是容器，但是该文件同样不能单独执行。Module 文件和 Component 文件作用一样，同样可以实现代码共享和重用，这里不再详细讲述。

13.1.2　使用 ActionScript 创建组件

在 Flex 中，所有的组件都是 ActionScript 类。例如 Button 组件，对应 Button.as 类文件。在 Flex 中，类分为组件类、管理类、数据服务类和其他用来实现 Flex 功能的类。这些类之间存在各种关系，包括继承等。图 13-4 展示了一些可视化组件类的结构。

从图 13-4 中可以看出，所有可视化组件最终都继承自 Sprite 类。该类是基本显示列表构造块：一个可显示图形、文字并且也可包含子项的显示列表节点。

Sprite 对象与影片剪辑类似，但没有时间轴。Sprite 类是 ActionScript 3.0 中新引入的类，提供了 MovieClip 类功能的替代功能。此替代功能保留了 ActionScript 以前版本的所有功能，

以提供后向兼容性。MovieClip 类在 ActionScript 3.0 中仍然存在，可以通过该对象创建时间轴动画。当然这些动画，在 Flash 软件中创建可能更方便。在 Flex 中，默认创建的可视化组件继承自 Sprite 类。下面开始在 Flex 中创建 ActionScript 组件。

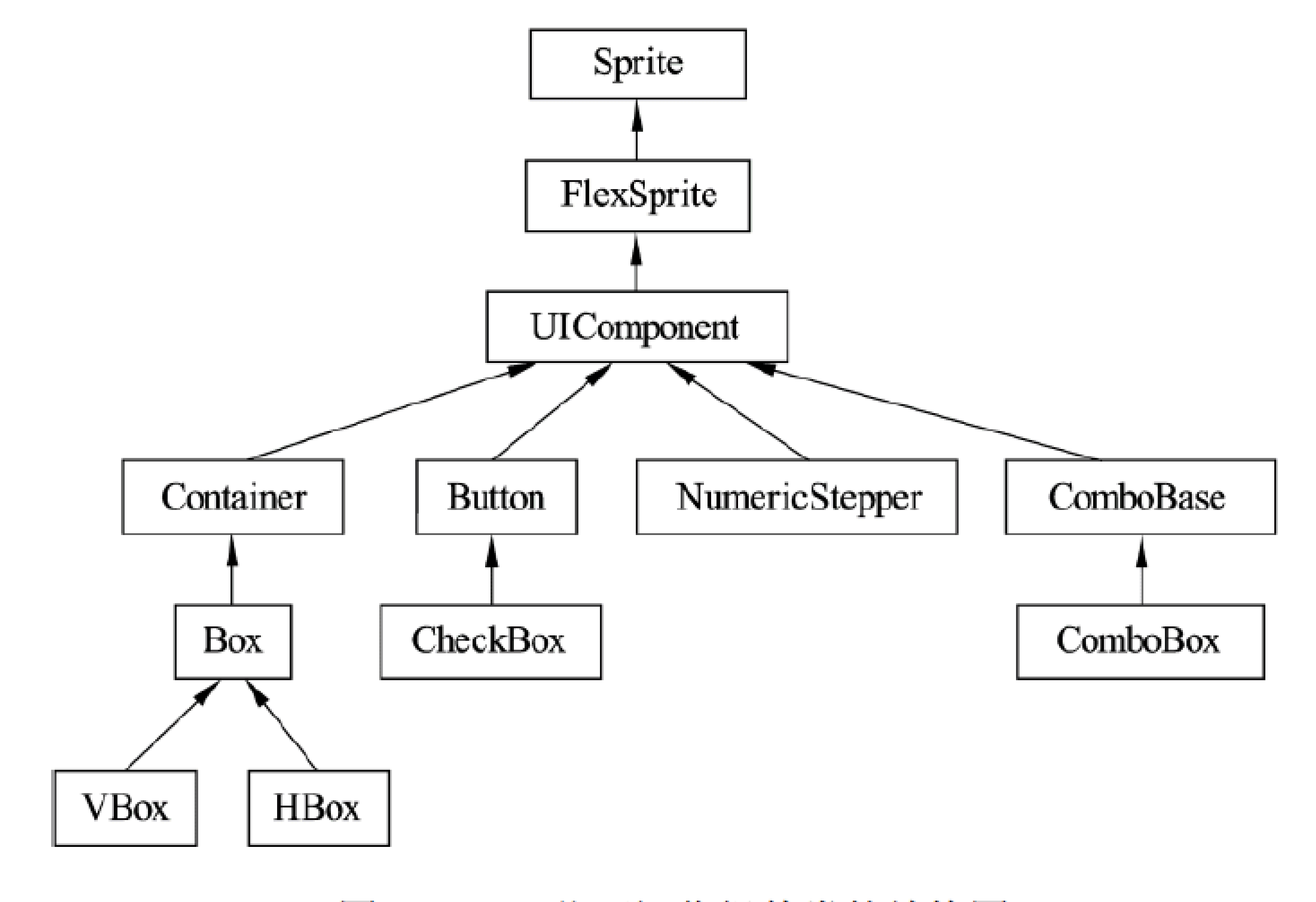


图 13-4　一些可视化组件类的结构图

首先选择 File|New|ActionScript Class 命令，弹出 New ActionScript Class 对话框，如图 13-5 所示。

在 Package 文本框中填写打包名，或者通过浏览找到要打包的文件夹，在 Name 文本框中填写类文件名。类文件名首字母大写，这是写作习惯。在 Modifiers 选择框中选择类修饰符和关键字。Superclass 文本框用来填写创建类的基类，也可以单击 Browse 按钮，从弹出的 Open Type 对话框中选择基类，如图 13-6 所示。Interfaces 用来添加实现的接口，可以通过单击 Add 按钮，打开 Open Type 对话框，选择可用来实现的接口。

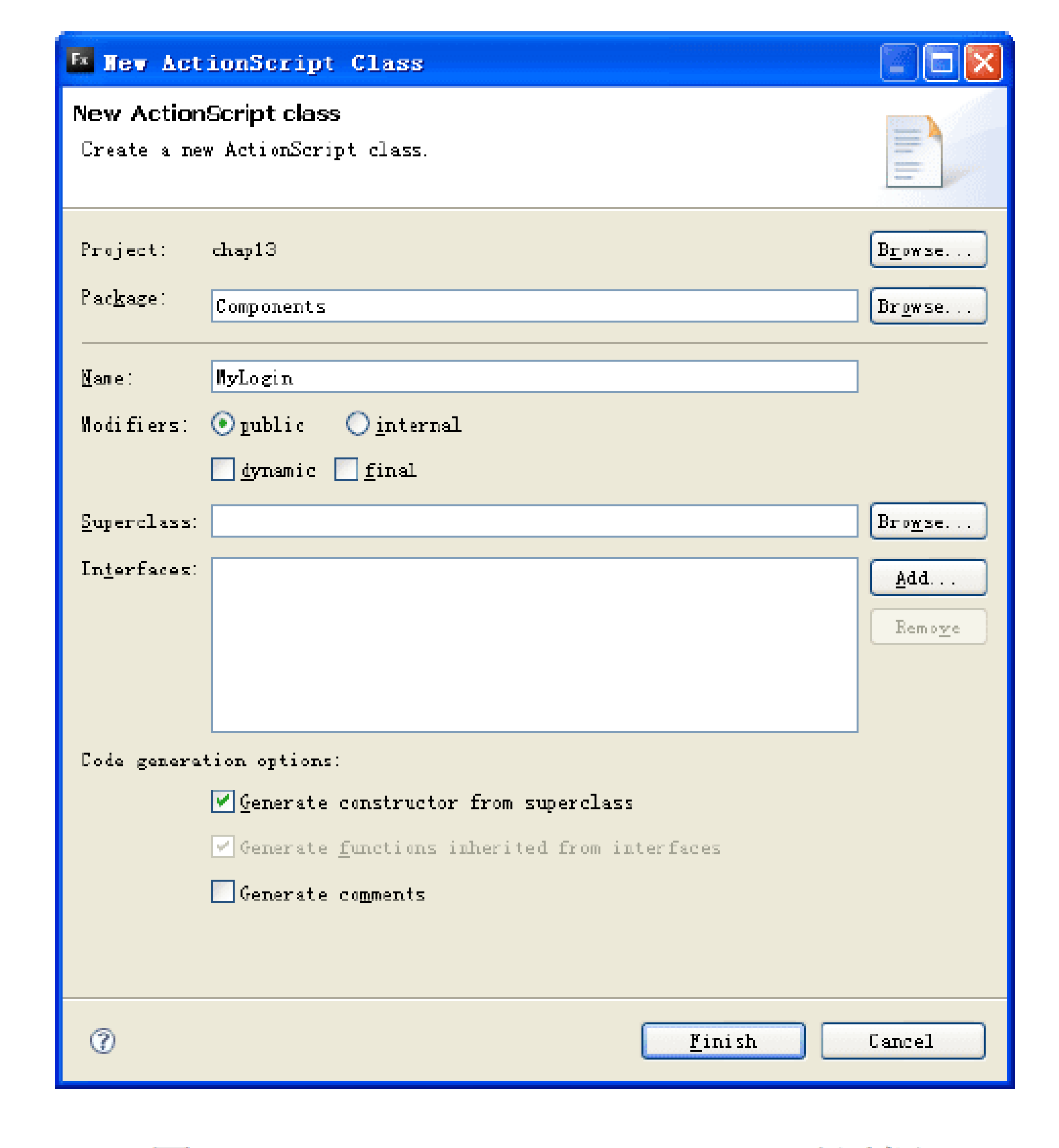


图 13-5　New ActionScript Class 对话框

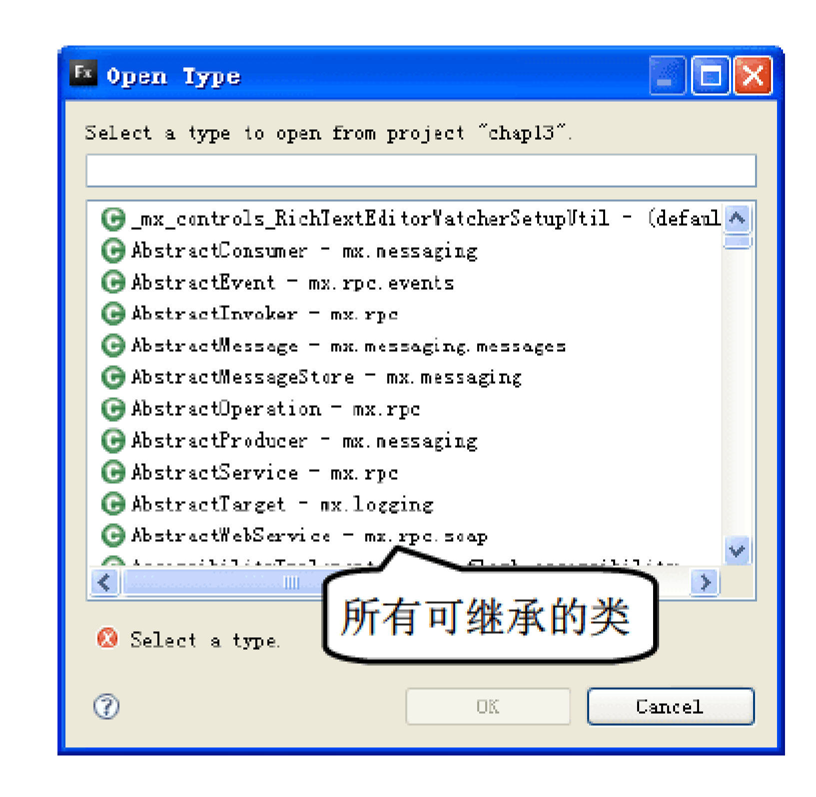


图 13-6　选择基类

Code generation options 复选框包括 3 项，其中第一项表示是否在创建类时，自动创建该类的构造函数，并且该构造函数调用基类构造函数；第二项表示是否创建实现接口的函数，该项只有在 Interfaces 中添加接口后才有效；第三项表示是否添加注释，方便以后阅读。

填写完成后，单击 Finish 按钮完成创建，这样在打包文件夹下，就会出现一个.as 后缀的文件，如图 13-7 所示，该文件的内容如代码 13.2 所示。

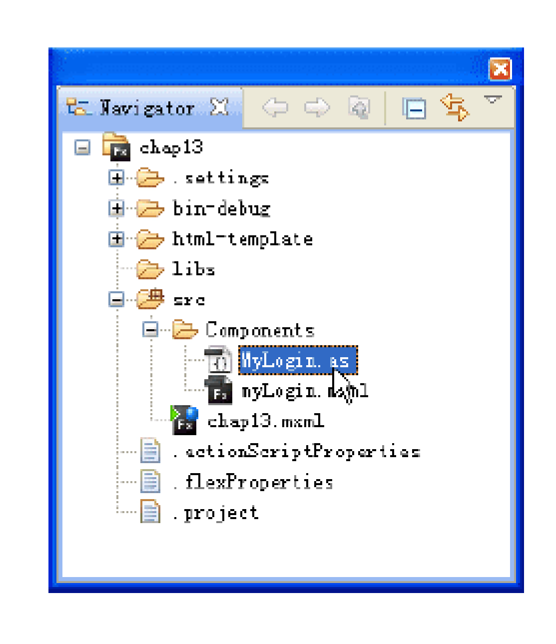


图 13-7　创建完成后生成的.as 文件

代码 13.2　创建的 ActionScript 类文件：MyLogin.as

```
package Components
{
    public class MyLogin extends TitleWindow
    {
        public function MyLogin(){
        }
    }
}
```

除了可以创建 ActionScript Class 文件，来创建组件外，还能通过创建 ActionScript File 文件。ActionScript File 是一个脚本文件，可以编写任意在 Flex 中用到的脚本，通过在主程序中调用即可。与 ActionScript Class 方式创建组件相比，ActionScript File 方式的包、类、构造函数等都需要程序员编写，其他的都一样。

13.2　在组件文件中添加项目

组件文件创建成功之后，就可以对该组件进行修饰。例如，对按钮组件的样式进行更改，在容器组件中添加新的组件等。有了这些功能，创建的组件就有了特色，而不仅仅是系统提供的样式。本节将要介绍如何在容器中添加项目以及 Flex 中样式的相关知识。

13.2.1　在 MXML 文件中添加项目

13.1 节介绍了两种在 Flex 中创建组件的方法，基本上所有能用 ActionScript 完成的自定义组件，都可以通过 MXML 来实现。一般来说，对于简单的自定义组件，例如修改已有组件的一些属性和方法，使用 MXML 要比使用 ActionScript 方便得多。当在新的组件中使用了别的组件，而且需要使用 Flex Layout 容器来进行多个组件的布局设置时，使用 MXML 来定义更方便。如果要修改某一个组件的行为，例如一个容器中子元素的布局方式，则使用 ActionScript。如果要通过创建 UIComponent 的子类来创建一个全新的可视化组件，则使用 ActionScript。如果要创建一个全新的非可视化组件，例如 Formatter，Validator，或者 Effect

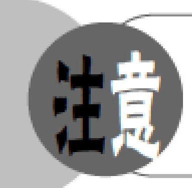
组件，则使用 ActionScript。



两者可以共存于一个项目中，但是不能在一个包内出现重名。

1. 静态添加

静态添加是在程序运行之前，就要把对象添加到场景中，完成属性设置。在 MXML 文件中添加组件和在主程序中添加组件一样，可以在 Design 视图下，将组件从 Components 面板中，拖曳到编辑区；也可以在 Source 视图下，输入以标识符"mx:"开头的标签，例如，<mx:Button>表示按钮组件。还可以在 Source 视图下添加样式和脚本，将在下面章节中介绍。



在非容器组件中，不可以添加组件，例如非容器组件 Button 等。

下面在第 13.1.1 小节创建的 myLogin.mxml 文件中，添加 2 个 Text 组件、2 个 TextInput 组件和 2 个 Button 组件，并调整组件的位置，得到如图 13-8 所示的效果。添加后文件内容如代码 13.3 所示。

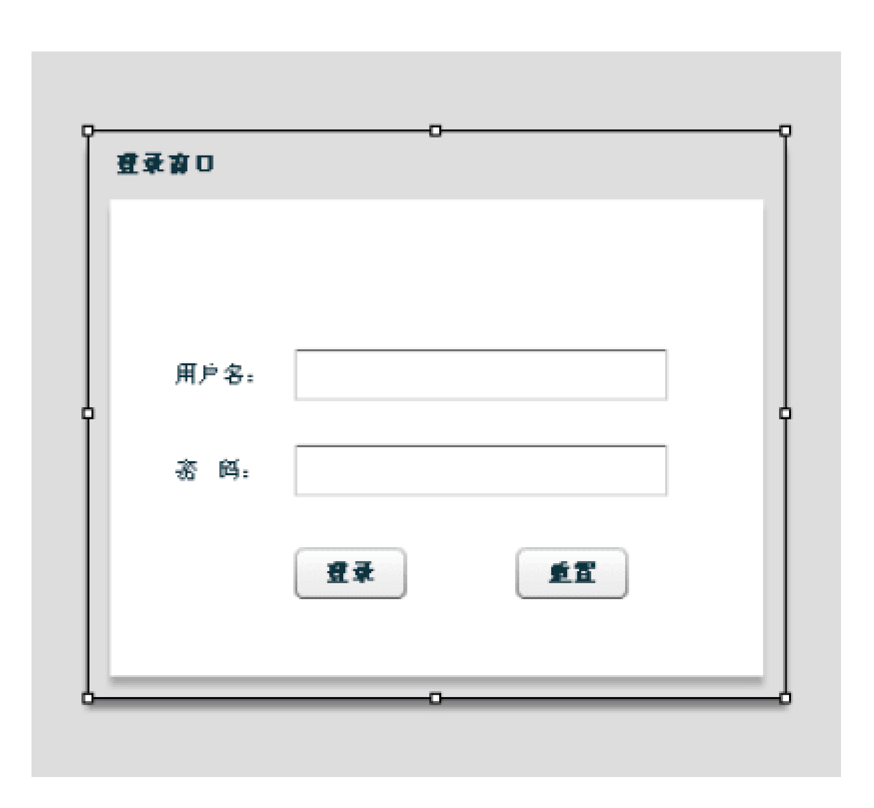


图 13-8　添加组件后的自定义组件效果

代码 13.3　在自定义组件中添加组件后代码

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="300" height="244" title="登录窗口">
    <mx:TextInput x="79" y="64" id="tiUserName"/>
    <mx:Text x="26" y="66" text="用户名: "/>
    <mx:TextInput x="79" y="105" id="tiUserPassword"/>
    <mx:Text x="26" y="107" text="密  码: "/>
    <mx:Button x="79" y="149" label="登录"/>
    <mx:Button x="174" y="149" label="重置"/>
</mx:TitleWindow>
```

2. 动态添加

动态添加是指在程序运行中，通过代码控制，将对象添加到场景中。这种情况有点类似下一节讲到的在 ActionScript 文件中添加项目，可以调用 addChild()和 addChildAt()方法添加，可以参考下一节。除了这种方法外，还可以通过调用 mx.manager 包中的弹出方法，向场景中添加对象。

mx.manager 包中有 addPopUp()和 centerPopUp()两个方法，可以实现弹出效果。其中 addPopUp()方法弹出一个可视化对象，该对象在场景中居于最上层，也就是下一节讲到的深度值最大。该方法的定义代码如下所示。

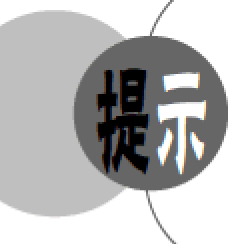
```
public static function addPopUp(window:IFlexDisplayObject,parent:
DisplayObject,
modal:Boolean = false,childList:String = null):void
```

该方法容许有 4 个参数，其中第一个参数 window 为需要弹出的可视化对象；第二个参数 parent 为弹出窗口的父对象，该对象容纳弹出窗口；第三个参数 modal 表示是否在弹出窗口后，不容许与其他对象进行交互，也就是该对象类似与警告框，弹出之后不容许对场景中的其他对象进行操作；第四个参数只能为 PopUpManagerChildList 枚举类的成员，表明弹出窗口的相对目标。

下面在代码 13.3 的基础上，添加一个按钮，当单击这个按钮时，弹出另外一个 TitleWindow 窗口。弹出窗口的代码如代码 13.4 所示，效果如图 13-9 所示。

代码 13.4　使用 addPopUp()方法弹出窗口

```
<mx:Script>
    <![CDATA[
        import mx.containers.TitleWindow;
        import mx.managers.PopUpManager;
        internal function popwin():void{
            var newwin:TitleWindow=new TitleWindow;
            newwin.title="弹出的窗口";
            newwin.width=200;
            newwin.height=200;
            PopUpManager.addPopUp(newwin,this,false);
        }
    ]]>
</mx:Script>
```



单击【弹出一个新窗口】按钮可以弹出若干个窗口，Flash Player 自动为每一个窗口指定 Name。弹出的对象如果带有 title 属性，一般情况下都是可以被拖动的。

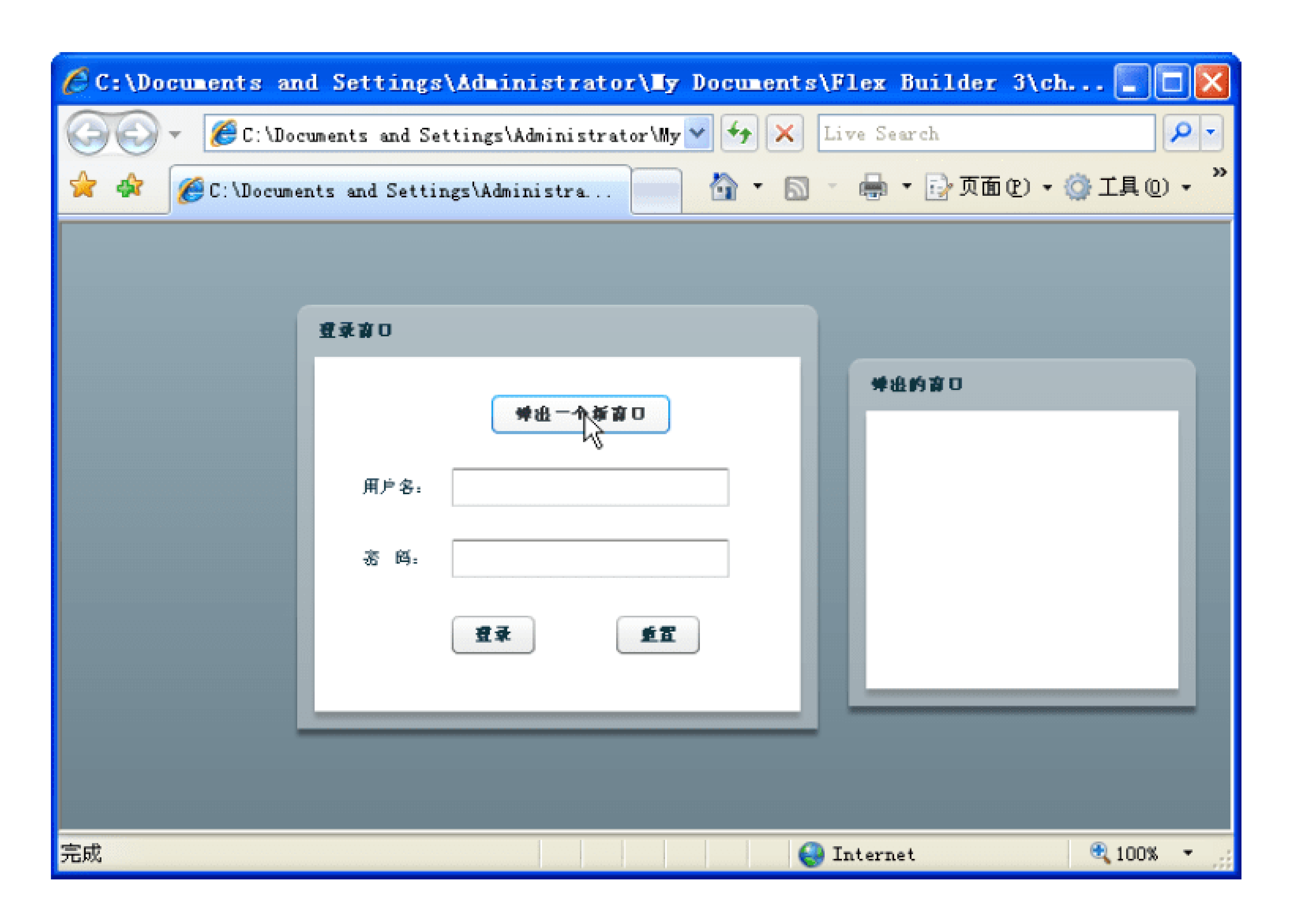


图 13-9　使用 addPopUp()方法弹出窗口效果

centerPopUp()方法与 addPopUp()方法相似，只是弹出的窗口在场景中位置居中。该方法只有一个参数，即需要弹出的可视化对象，但是该对象必须是通过调用 createPopUp()方法创建的弹出对象。centerPopUp()方法的定义代码如下所示。

```
public static function centerPopUp(popUp:IFlexDisplayObject):void
```

下面在主程序中添加一个按钮，单击该按钮，弹出已创建的 myLogin 组件，并且该组件的位置在场景中居中。主程序的内容如代码 13.5 所示，效果如图 13-10 所示。

代码 13.5　使用 centerPopUp()方法弹出窗口

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
xmlns:ns1="Components.*">
    <mx:Button x="20" y="24" label="弹出居中窗口" click="showWin()"/>
    <mx:Script>
        <![CDATA[
            import Components.myLogin;
            import mx.managers.PopUpManager;
            internal function showWin():void{
                PopUpManager.centerPopUp(PopUpManager.createPopUp(this,
                myLogin));
            }
        ]]>
    </mx:Script>
</mx:Application>
```

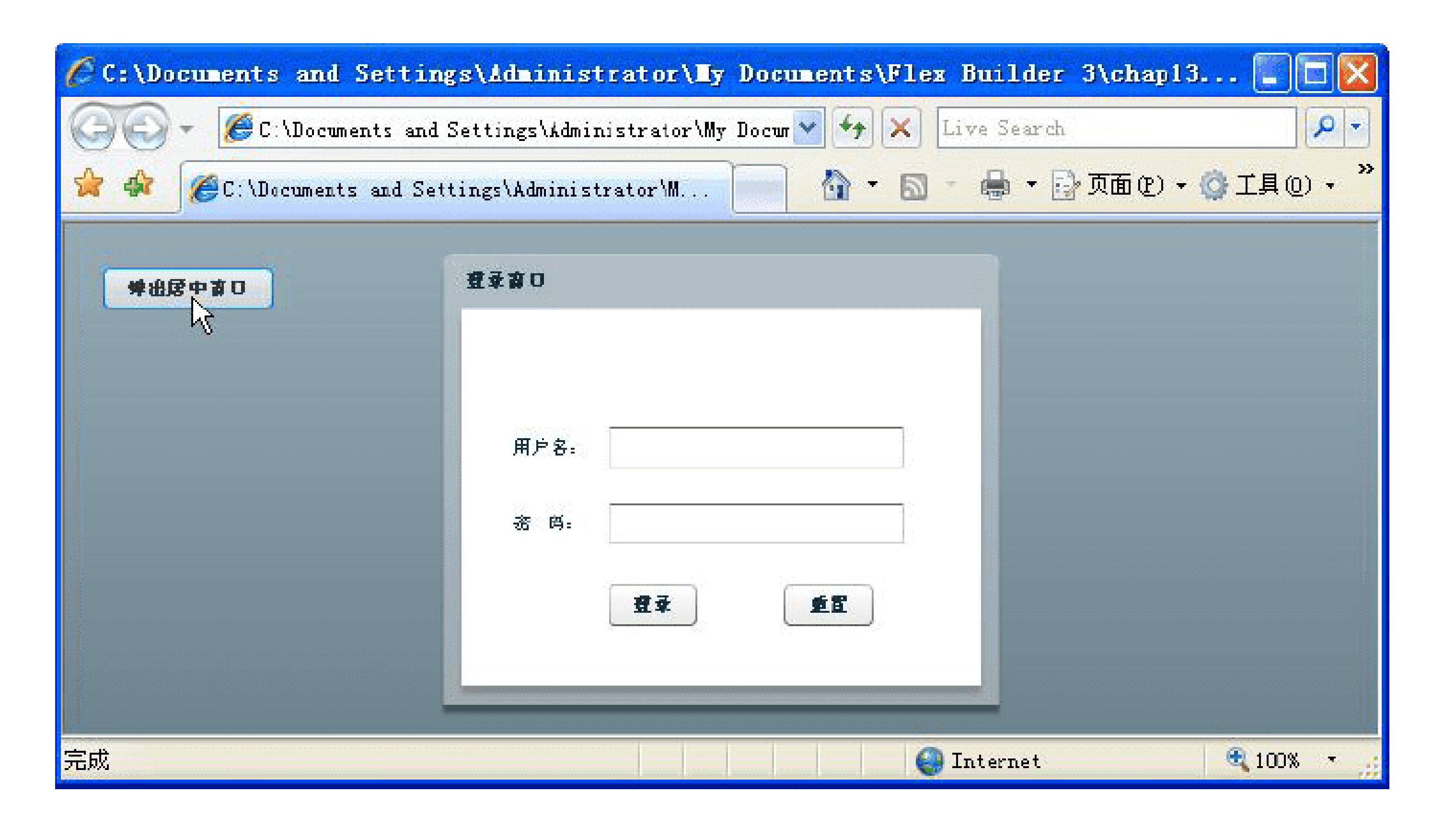


图 13-10　使用 centerPopUp()方法弹出窗口效果

13.2.2　在 ActionScript 文件中添加项目

在 Flash Player 中，由 ActionScript 虚拟机（AVM）和渲染引擎实现在场景中显示对象。AVM 负责执行 ActionScript 代码、创建对象，渲染引擎负责在屏幕上绘制对象。其中创建对象需要用到 new 操作符，实例化组件的类。例如创建一个 Button 对象的代码如下所示。

```
var myButton:Button=new Button;
```

上述代码在 AVM 中创建了一个 Button 对象，但是该对象没有绘制到屏幕上，因为它还不存在于渲染引擎中。要把它放到渲染引擎中，需要添加该对象到可视化对象容器中，通过调用 addChild()和 addChildAt()方法添加。

1. addChild()方法

addChild()方法将一个 DisplayObject 对象，添加到该 DisplayObjectContainer 对象中，该方法的定义语句如下所示。

```
override public function addChild(child:DisplayObject):DisplayObject
```

当在 DisplayObjectContainer 对象中添加对象时，添加的对象就成为 DisplayObjectContainer 对象的子对象。这样就有了一定的级别结构，其中 stage 是所有可视化对象的根节点。

在 13.1.1 节创建的 TitleWindow 组件中，添加【登录】按钮可以使用如代码 13.6 所示的代码。

代码 13.6　在容器中添加按钮组件

```
package Components
{
    import mx.containers.TitleWindow;
    import mx.controls.Button;
    public class MyLogin extends TitleWindow
```

```
        {
            public function MyLogin(){
                var myButton:Button=new Button;
                myButton.label="登录";
                addChild(myButton);
            }

        }
    }
```

2. addChildAt()方法

对象添加到可视化容器中，还具有一定的深度关系，这种关系决定着对象在屏幕上显示的顺序。一般情况下，后添加的对象所处的深度靠上。在 Flex 中，每个对象在容器中都有个深度位置，用一个整数来索引。位置 0 代表最底层，处在该位置的对象在处在位置 1 的对象下绘制，位置 1 在位置 2 下绘制，依次类推。使用过 Flash 和 Photoshop 的用户，对深度（或图层）概念比较容易接受。

当通过 addChild()方法添加对象时，该对象会显示在最顶层，也就是该对象得到最大的深度索引值。若要自己指定对象的深度，就要用到 addChildAt()方法，该方法的定义语句如下所示。

```
override public function addChildAt(child:DisplayObject, index:int):
DisplayObject
```

下面通过一个实例，演示如何使用上述两个方法。首先在 Flex 中创建一个 ActionScript Project，系统自动生成一个主文件，该文件继承自 Sprite 类。在此文件中编写代码，创建 3 个不同颜色的正方形，使用 addChild()方法添加到场景中。再创建同样的 3 个正方形，只是位置不同，使用 addChildAt()方法添加到场景中。具体内容如代码 13.7 所示，效果如图 13-11 所示。

代码 13.7　使用 addChild()和 addChildAt()方法添加对象

```
package {
    import flash.display.Shape;
    import flash.display.Sprite;
    public class chap13as extends Sprite
    {
        public function chap13as()
        {
            stage.scaleMode="noScale";
            //创建 3 个正方形
            var greenRect:Shape=CreateRect(0x00ff00,50,50,100,100);
            var redRect:Shape=CreateRect(0xff0000,50,60,100,100);
            var blueRect:Shape=CreateRect(0x0000ff,50,70,100,100);
            //使用 addChild()方法添加到容器
```



```
                addChild(greenRect);
                addChild(redRect);
                addChild(blueRect);
                var greenRect1:Shape=CreateRect(0x00ff00,200,50,100,100);
                var redRect1:Shape=CreateRect(0xff0000,200,60,100,100);
                var blueRect1:Shape=CreateRect(0x0000ff,200,70,100,100);
                addChildAt(greenRect1,3);
                addChildAt(redRect1,2);
                addChildAt(blueRect1,1);
            }
            //创建函数，用来显示正方形
            protected function CreateRect(color:uint,x:Number,y:Number,
            width:Number,height:Number):Shape{
                var myshap:Shape=new Shape;
                myshap.graphics.beginFill(color,1);
                myshap.graphics.drawRect(x,y,width,height);
                myshap.graphics.endFill();
                return myshap;
            }
        }
    }
```

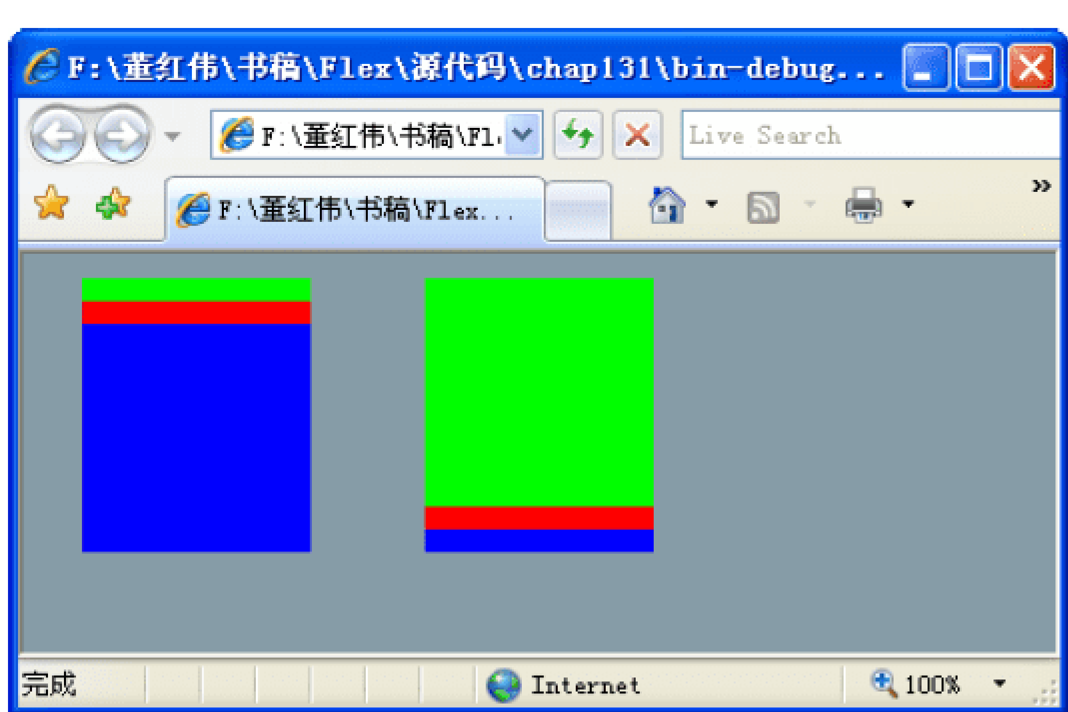


图 13-11　使用 addChild()和 addChildAt()方法添加对象的不同效果

13.3　使用 CSS 样式

无论是网页还是程序，都需要对界面进行美化。系统自带的组件界面虽然已经比较漂亮，但是还是满足不了视觉上的要求。这就需要使用 CSS 样式，使得组件具有一定的视觉效果，融合到整个网站或程序中。Flex 比 Flash 更适合网页开发，优势之一就是 Flex 支持的 CSS 样式要比 Flash 丰富，样式定义的方法也很多。本节将介绍 Flex 中样式的基础知识和如何使用样式。

13.3.1　CSS 样式语法

样式表由多条样式规则组成，每种样式规则都设置一种样式。一种样式规则，就是针对对象所设定的显示样式。样式规则的定义语法非常简单，都是由一些属性标签组成的。

1．基本语法

样式表是由若干条样式规则组成的，这些样式规则可以应用到不同的元素或文档来定义它们显示的外观。每一条样式规则由 3 部分构成：选择符（Selector）、属性（properties）和属性的取值（value）。其基本格式如下：

```
Selector{property: value}
```

Selector 选择符必须为 Flex 中的可视化组件或可视化自定义组件，如 Button、Text、myLogin（自定义组件）等。property 属性则是选择符指定的标签所包含的属性。value 指定了属性的值。如果定义选择符的多个属性，则属性和属性值为一组，组与组之间用分号（;）隔开。格式如下：

```
Selector{property1: value1; property2: value2;…… }
```

下面就给出一条样式规则，如下所示。

```
Button {color: #ff0000;}
```

该样式规则中 Button 是指为按钮提供样式，color 为指定文字颜色的属性，#ff0000 为属性值，表示按钮的文字颜色为红色。

如果属性值由多个值组成，那么该属性值就必须使用逗号（,）隔开。例如，设置 Panel 的标题颜色由白色过渡到深灰色，样式规则如下：

```
Panel {headerColors: #ffffff, #666666;}
```

如果属性值为字符串类型，那么该属性值就必须使用双引号（""）隔开。例如，设置 Panel 的标题样式名为 mypanelTitle，样式规则如下：

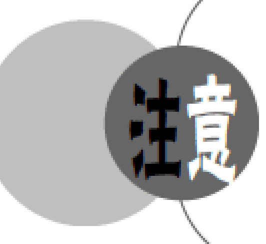
```
Panel {titleStyleName: "mypanelTitle";}
```

如果要为 Panel 同时设置多种样式，则可以使用下列语句。

```
Panel {borderColor: #000000; headerColors: #ffffff, #666666;titleStyleName:
"mypanelTitle";}
```

一般情况下，为了便于阅读，书写样式规则时，可以采用分行的格式，例如：

```
Panel {
   borderColor: #000000;
   headerColors: #ffffff, #666666;
   titleStyleName: "mypanelTitle";
}
```



Flex 中，并非所有组件的可视化属性都能作为样式表的属性，例如 UIComponent 类的 x、y、width 和 height 并不是样式属性，当然也不能在 CSS 文件中设定。

2. 选择符集合

有些标签的样式可以是相同的，例如颜色、大小。这时就可以将具有相同属性和值的选择符组合起来形成选择符组统一定义，选择符之间用逗号（,）隔开，这样可以减少样式的重复定义。

例如，设定按钮和文字区域组件的字体大小为 14，颜色为红色。

```
Button, TextArea {fontSize: 14; color: #ff0000;}
```

该样式规则还可以写为：

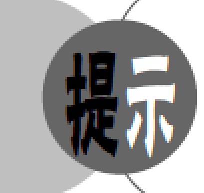
```
Button {fontSize: 14; color: #ff0000;}
TextArea {fontSize: 14; color: #ff0000;}
```

3. 类选择符

使用类选择符能够为相同的标签定义不同的样式，也可以把相同的样式使用到不同的标签上。定义类选择符时，需要在自定义类的名称前面加一个句点（.）。例如，为 Button 组件定义两个类来表示不同的样式。

```
Button.red{color: #ff0000;}
Button.green{color:#00ff00;}
```

上面两个样式规则，Button 表示样式应用的组件为按钮，red、green 为定义的类选择符的类的名称，{}内为样式定义。



类的名称可以是任意英文字符串或以英文开头的英文字符与数字的组合，一般情况下，是以其功能及效果的简写作为名称。

将定义的类选择符应用到不同的 Button 组件中，只要在<mx:Button/>标签中指定 styleName 属性即可。

```
<mx:Button styleName ="red" label="红色"/>
<mx:Button styleName ="green" label="绿色"/>
```



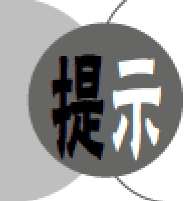
一个组件对象只能使用一个类选择符定义的样式。

上面定义的类选择符只适用于一种标签，当然也可以将定义类选择符应用于具有相同样式的不同标签。此时，类选择符中句点（.）前就可以将标签名省略。例如：

```
.red{color:#ff0000}
```

该样式规则表示定义.red 的类选择符颜色显示为红色，可以被应用于所有具有 color 属性的标签，从而实现相同的样式外观。例如：

```
<mx:Button label="登录" styleName ="red"/>
<mx:TextArea styleName ="red" text="使用样式"/>
```

定义省略标签的类选择符，可以很方便地在任意组件对象上应用预先定义好的类样式。

13.3.2　创建 CSS 文件

在 Flex 软件中，为用户创建 CSS 样式提供了界面化的操作环境，实现了所见即所得，使开发样式变得简单。本小节讲述如何使用 Flex 环境开发样式。

首先在 Flex 中新建一个 CSS 文件。通过选择 File|New|CSS File 命令，打开新建 CSS 文件的窗格，如图 13-12 所示。选择 src 文件夹为存放文件的位置，在 Filename 文本框中输入 CSS 文件的文件名称，单击 Finish 按钮完成创建。

创建好的 CSS 文件打开后，编辑区的效果如图 13-13 所示。通过观察可以看出，该文件也是可以转到设计模式的，也就是可以预览的。

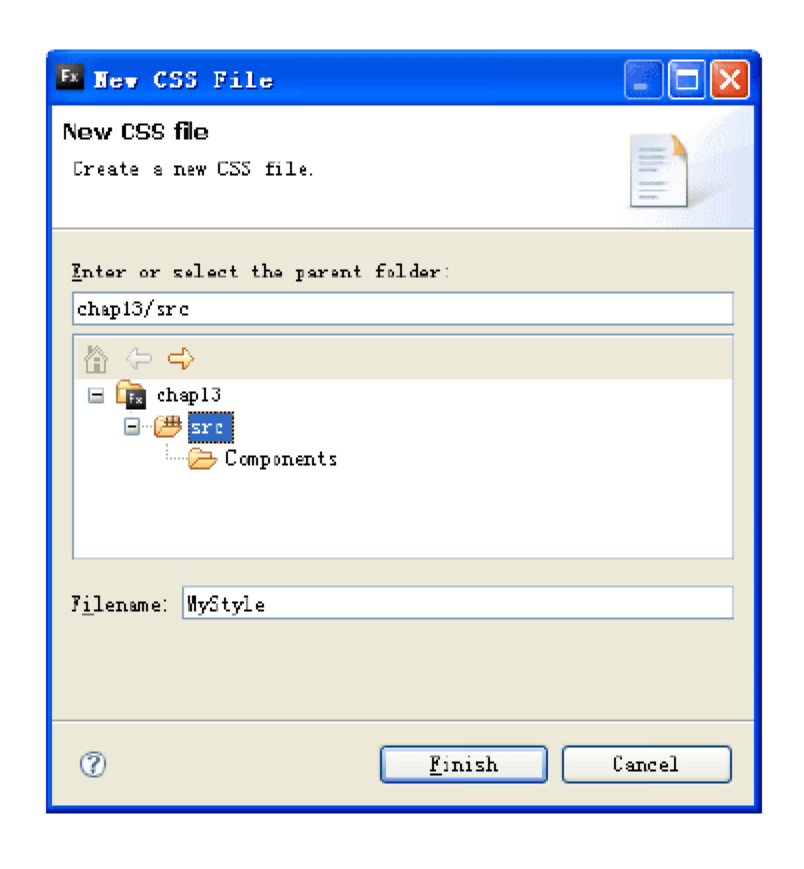


图 13-12　新建 CSS 文件窗格效果



图 13-13　CSS 文件在编辑区的效果

在设计模式下，可以通过单击【添加样式】按钮添加新样式，弹出新建样式对话框。在该对话框中可以创建样式规则，窗格效果如图 13-14 所示。

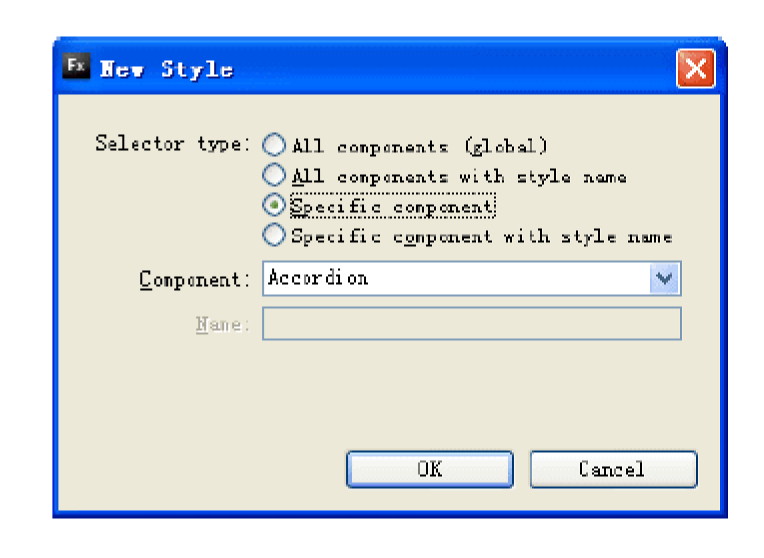


图 13-14　新建样式窗格

在新建样式对话框中，需要选择 Selector type，即选择符类型，包括 4 种，如下所示。

- **All components（global）**　针对整个场景，定义名为 global 的全局样式。该样式不需要引用，会用到整个场景中。一个 CSS 文件中只能有一个 global

定义。

- **All components with style name**　创建可以被所有组件使用的样式。
- **Specific component**　针对特定组件开发样式。
- **Specific component with style name**　创建可以被特定组件使用的样式。

Component 下拉列表框，用来选择开发样式针对的组件，这些组件为 Flex 可视化组件或用户可视化组件。

Name 文本框用来定义样式名称。样式名称不要重复，否则有些样式将被覆盖。这一项只是在需要输入时才有效。

当选择 All components 选项后，单击 OK 按钮，创建 global 全局样式。初次创建时，会弹出 Select Component for Preview 对话框，如图 13-15 所示。该对话框用来选择创建样式的预览组件。例如，当选择 Button 组件，单击 OK 按钮之后，编辑区会显示按钮的所有样式，如图 13-16 所示。

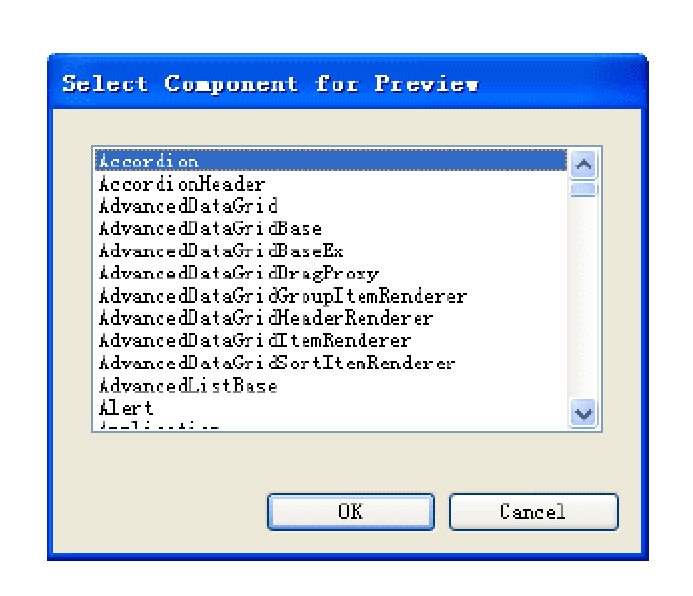


图 13-15　选择预览组件

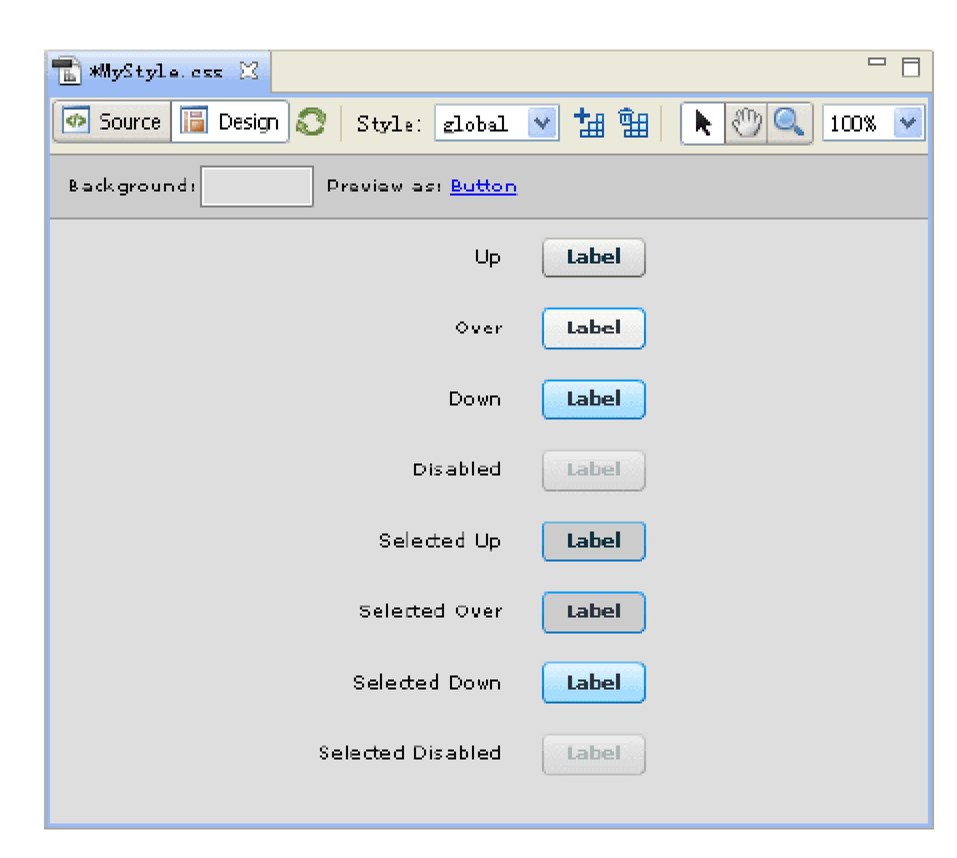
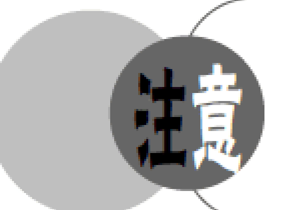


图 13-16　选择 Button 组件作为预览

在 Flex Properties 对话框中，显示对应 Button 组件的相关属性，可以进行修改，编辑区会随之改变效果，例如，Text 属性中改变文字颜色等。



global 样式的更改，不仅仅是改变某一组件的样式，而是对场景中所有组件都有效。

除了在预览模式下定义样式外，也可以在代码模式下，通过输入代码的形式，编写样式，这种方式对样式语法要求较高。

13.3.3　引用 CSS 样式

上一小节中，介绍了如何在 Flex 中创建 CSS 文件，该文件实际上就是个文本文件，也可以使用文本编辑器编写。在 Flex 中，除了创建 CSS 文件，还有其他方式为组件定义样式。本小节讲述如何在场景文件中使用 CSS 样式。

在 Flex 中，为组件添加样式的方法有以下 5 种。

- 使用本地样式
- 使用外部样式表
- 使用内联样式
- 使用 setStyle()方法
- 使用 StyleManager 类

1．使用本地样式

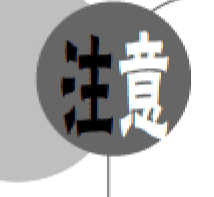
在 MXML 文件中，可以使用<mx:Style>标签创建本地样式定义。这些定义会应用到当前文档，以及当前文档的所有子文档。可以通过在 Flex Properties 对话框中进行设定，也可以在代码中添加。

例如，定义一个页面中两种字体。可以创建两个不同的类选择器，定义不同的字体，并将它们赋给两个 TextArea。完整内容如代码 13.8 所示，生成效果如图 13-17 所示。

代码 13.8　使用本地样式

```
    <?xml version="1.0" encoding="utf-8"?>
    <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
xmlns:ns1="Components.*">
    <mx:Style>
        .bigAndred{
            color: #ff0000;
            fontSize: 20;
        }
        .smallAndblue{
            color: #0051ff;
            fontSize: 14;
        }
    </mx:Style>
        <mx:TextArea x="34" y="58" width="381" height="180" styleName=
        "smallAndblue">
            <mx:text>在程序开发中…</mx:text>
        </mx:TextArea>
        <mx:TextArea x="34" y="10" text="Flex 教程" styleName="bigAndred" width=
        "97" height="40"/>
    </mx:Application>
```

如上例所示，使用本地样式，可以定义若干类选择符。通过类选择符，可以使同一个组件的不同实例，达到不同效果。



在自定义组件中定义的样式，使用组件作为选择符定义样式，在程序文件中不会起作用。而应该使用类选择符定义，再为组件对象设置 styleName 属性，或使用内联方式定义样式。

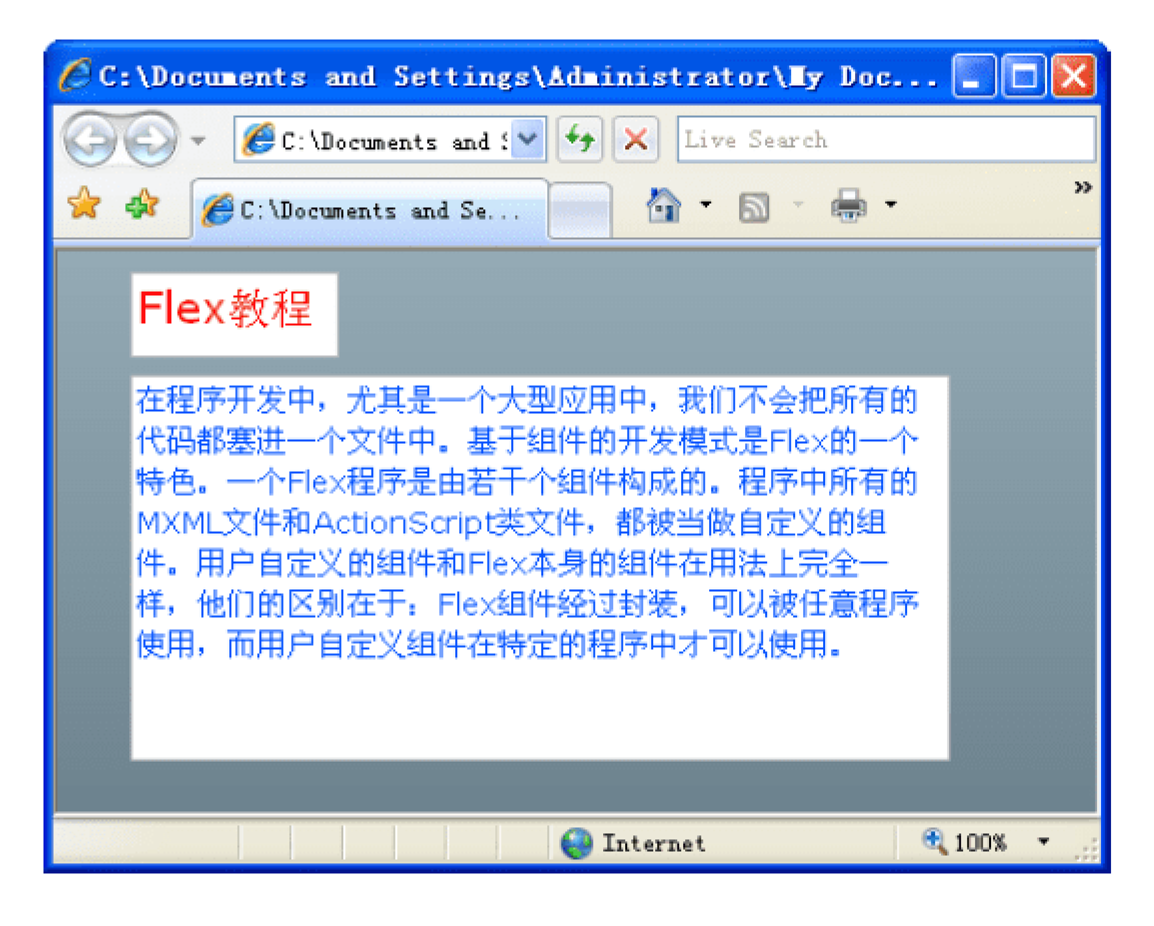


图 13-17　定义不同字体效果

2. 使用外部样式表

如果定义的样式不是太多，可以使用本地样式表定义。如果定义的样式较多时，程序文件就会很大，阅读起来不方便。用户可以将这些样式单独保存在一个.css 文件中，通过程序文件调用来使用样式。在 Flex 中，通过在<mx:Style>标签的 source 属性中指定外部 CSS 文件。

下面通过一个例子，看看如何使用外部样式表。首先在 Flex 中创建一个 CSS 文件，命名为 mystyle.css。在此文件中定义 HscrollBar 和 VscrollBar 样式，这两个选择符是 Flex 组件，分别代表横向和纵向滚动条。内容如代码 13.9 所示。

代码 13.9　定义横、纵向滚动条样式：mystyle.css

```
HScrollBar,VScrollBar {
    cornerRadius: 4;
    highlightAlphas: 0.7, 0;
    fillAlphas: 0.64, 0.4, 0.75, 0.65;
    fillColors: #ffffff, #0000ff, #ffffff, #eeeeee;
    trackColors: #ffffff, #e7e7e7;
    themeColor: #0000ff;
    borderColor: #00ff00;
}
```

然后在程序页面中引入上面创建的 CSS 文件。在程序文件的<mx:Application>标签内，添加<mx:Style>标签，并指定 source 属性为 CSS 文件的地址。

再在程序文件中，添加一个 Image 组件，并为其指定 Source 属性。该图片较大，当改变程序页面的大小时，页面会出现滚动条。程序文件的内容如代码 13.10 所示。

代码 13.10　程序文件代码

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
<mx:Style source="mystyle.css">
```

```
</mx:Style>
    <mx:Image x="0" y="0" source="images/view.jpg"/>
</mx:Application>
```

运行该程序，通过鼠标拖动，改变页面的大小，使页面产生滚动条，效果如图 13-18 所示。

3. 使用内联样式



图 13-18　改变横纵滚动条样式效果

所谓内联样式，是指直接设定页面中某个对象的属性，可以在预览模式或代码模式下设定。在预览模式下，可以在 Flex Properties 对话框中，设定对象属性。在代码模式下，可以通过添加代码设置。Flex 软件提供了智能提示功能，当在某一标签内按下空格键时，弹出提示框，显示出该组件支持的所有属性和方法，如图 13-19 所示。当输入某个字母时，提示框中内容进行筛选，提出以此字母开头的属性和方法。继续输入需要属性的字母，直到看到提示框中选中该属性，按回车键确认。

通过内联方式定义的样式，优先级要比本地样式和外部样式高，但是这种样式定义只对单个对象，局限性较大。下面通过为 RadioButton 组件定义样式，来看如何使用内联样式。

首先在程序文件中添加 5 个 RadioButton 组件，设置所有的 groupName 属性为 favorites，每一个的 label 属性不同。然后使用本地样式，为 RadioButton 组件定义类选择符样式，包括字体颜色和大小。最后，单独为最后一个 RadioButton 对象设置不同字体颜色。该程序文件的完整内容如代码 13.11 所示，运行该程序，效果如图 13-20 所示。

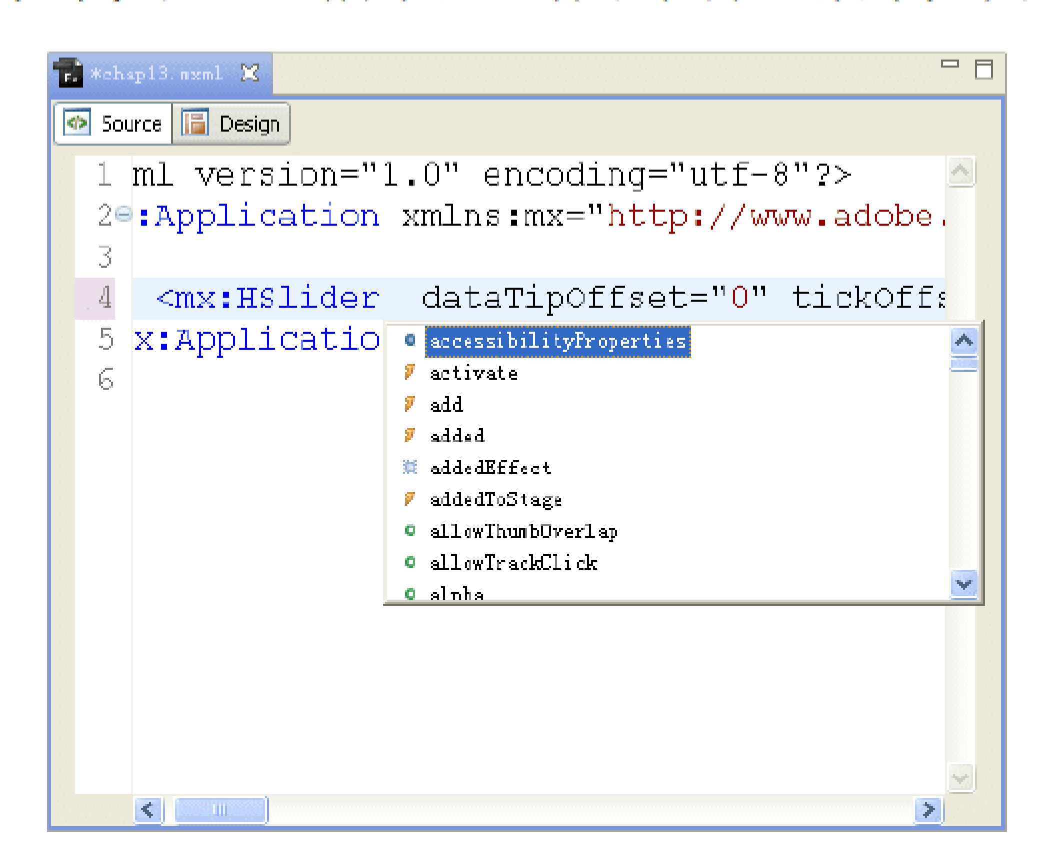


图 13-19　代码模式下弹出提示框效果



图 13-20　使用内联样式效果

代码 13.11　使用内联样式

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Style>
        RadioButton{
            color:#00ff00;
```

```
            fontSize:16px;
        }
    </mx:Style>
    <mx:RadioButton x="52" y="44" label="篮球" selected="true" groupName=
    "favorites"/>
    <mx:RadioButton x="52" y="69" label="足球" selected="false" groupName=
    "favorites"/>
    <mx:RadioButton x="52" y="97" label="排球" selected="false" groupName=
    "favorites"/>
    <mx:RadioButton x="52" y="123" label="羽毛球" selected="false" groupName=
    "favorites"/>
    <mx:RadioButton x="52" y="148" label="看书" selected="false" groupName=
    "favorites" color="#FF0C00"/>
</mx:Application>
```

4. 使用 setStyle()方法

上文中讲到的 3 种引用样式方法，是在程序编译时指定的，也可以在程序运行时动态指定，这需要使用 ActionScript 中的 setStyle()方法。使用这种方式，不能得到“所见即所得”的效果，需要程序员有较强的客户端处理能力。但是这种方式，为样式的应用提供了更加灵活的功能。

setStyle()方法有两个参数：样式名称和样式的值，该方法的定义语句如下所示。

```
public function setStyle(styleProp:String, newValue:
*):void
```

下面通过一个实例来演示如何使用 setStyle()方法。该实例使用 ColorPicker 组件，通过选择颜色，改变场景背景颜色的功能。

首先创建一个程序文件，在此文件中添加一个 ColorPicker 组件，命名为 cpBgColor，change 属性设置为 ChangeBg()。ChangeBg()函数，可以在文件中定义，主要是根据 cpBgColor 对象选中的颜色值，更改场景的背景颜色。程序文件的内容如代码 13.12 所示，运行后，效果如图 13-21 所示。

代码 13.12　使用 setStyle()方法更改场景的背景色

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Script>
        <![CDATA[
            internal function ChangeBg():void{
                this.setStyle("backgroundColor",cpBgColor.value);
            }
        ]]>
    </mx:Script>
    <mx:ColorPicker x="122" y="10" change="ChangeBg()" id="cpBgColor"/>
    <mx:Text x="10" y="10" text="更改背景颜色: " fontSize="16"/>
```

```
</mx:Application>
```

5. 使用 StyleManager 类

在 Flex 中，提供了管理样式的 StyleManager 类，该类位于 mx.styles 包中。在程序文件中，通过创建样式对象（CSSStyleDeclaration），为对象设置新的样式属性。然后通过 StyleManager 对象的 setStyleDeclaration()方法，将样式对象添加到页面中。setStyleDeclaration()方法的定义语句如下所示。

```
public static function setStyleDeclaration
(selector:String,styleDeclaration:
CSSStyleDeclaration,
update:Boolean):void
```

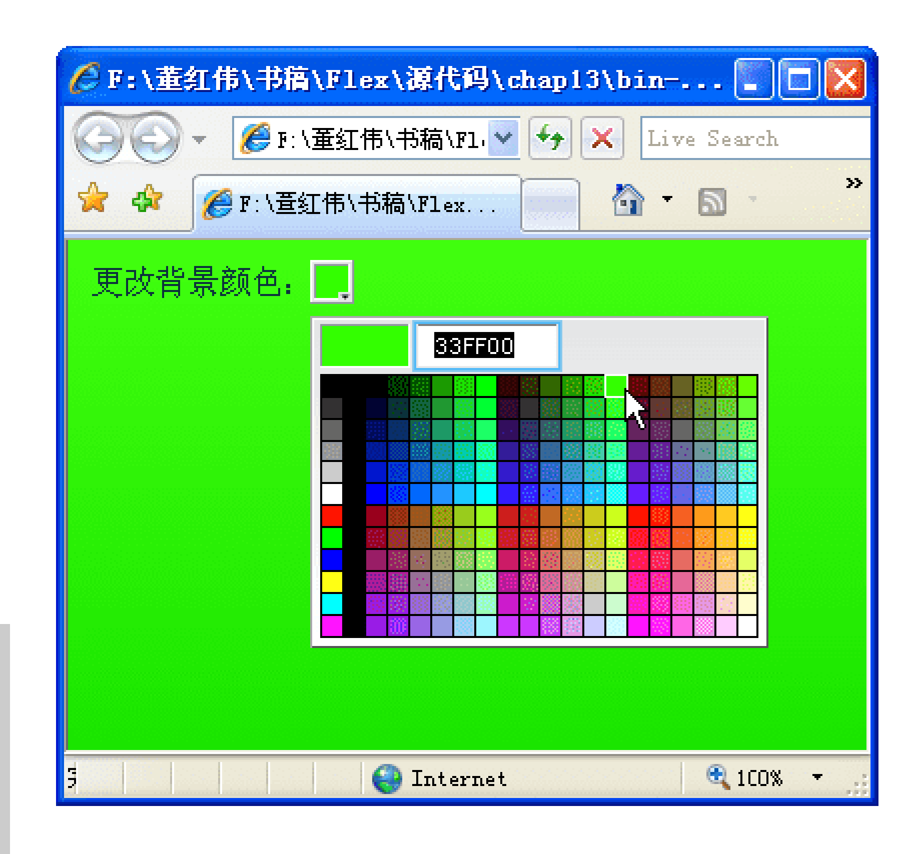


图 13-21　选择绿色时场景背景色效果

参数 selector 表示新的样式选择符，styleDeclaration 为新的样式对象，update 表示是否立即生效。当 update 为 true 时，表示新的样式立即生效，应用到所有对应的目标对象。如果为 false，不会立即生效，直到再次调用 StyleManager 的 clearStyleDeclaration()方法时才起作用。

下面通过一个实例演示如何使用 StyleManager 类，定义样式。该实例可以根据用户的选择，设置文本框中文字的颜色和大小。

首先创建程序文件，在该文件中添加 1 个 ColorPicker 组件和 1 个 NumericStepper 组件，ColorPicker 组件用来选择文字颜色，NumericStepper 组件用来设置大小。再添加 1 个 TextArea 组件和一个 Button 组件，分别用来显示文字和提交修改。

然后设置组件属性，主要是为 ColorPicker 和 NumericStepper 组件指定 id 以及为 TextArea 组件添加内容，为 Button 组件添加 click 方法。最终程序文件内容如代码 13.13 所示，运行后效果如图 13-22 所示。

代码 13.13　使用 StyleManager 类更改文字样式

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Script>
        <![CDATA[
            import mx.styles.StyleManager;
            internal function ChangeStyle():void{
                //定义样式对象
                var myTextArea:CSSStyleDeclaration=new CSSStyleDecla-
                tion();
                //设置样式属性
                myTextArea.setStyle("color",cpColor.value);
                myTextArea.setStyle("fontSize",nsFontSize.value);
                //添加到页面，并立即生效
                StyleManager.setStyleDeclaration(".myStyle",myTextArea,
```

```
                    true);
                }
            ]]>
        </mx:Script>
        <mx:ColorPicker x="67" y="10" id="cpColor"/>
        <mx:Text x="10" y="10" text="字体颜色: "/>
        <mx:Text x="99" y="10" text="字体大小: "/>
        <mx:NumericStepper x="162" y="8" id="nsFontSize" value="12" minimum="1"
        maximum="30" stepSize="2"/>
        <mx:TextArea x="10" y="36" width="504" height="192" styleName="myStyle">
            <mx:text>郑州汇智计算机技术服务有限公司成立于 2005 年…</mx:text>
        </mx:TextArea>
        <mx:Button x="242" y="8" label="更改" click="ChangeStyle()"/>
    </mx:Application>
```

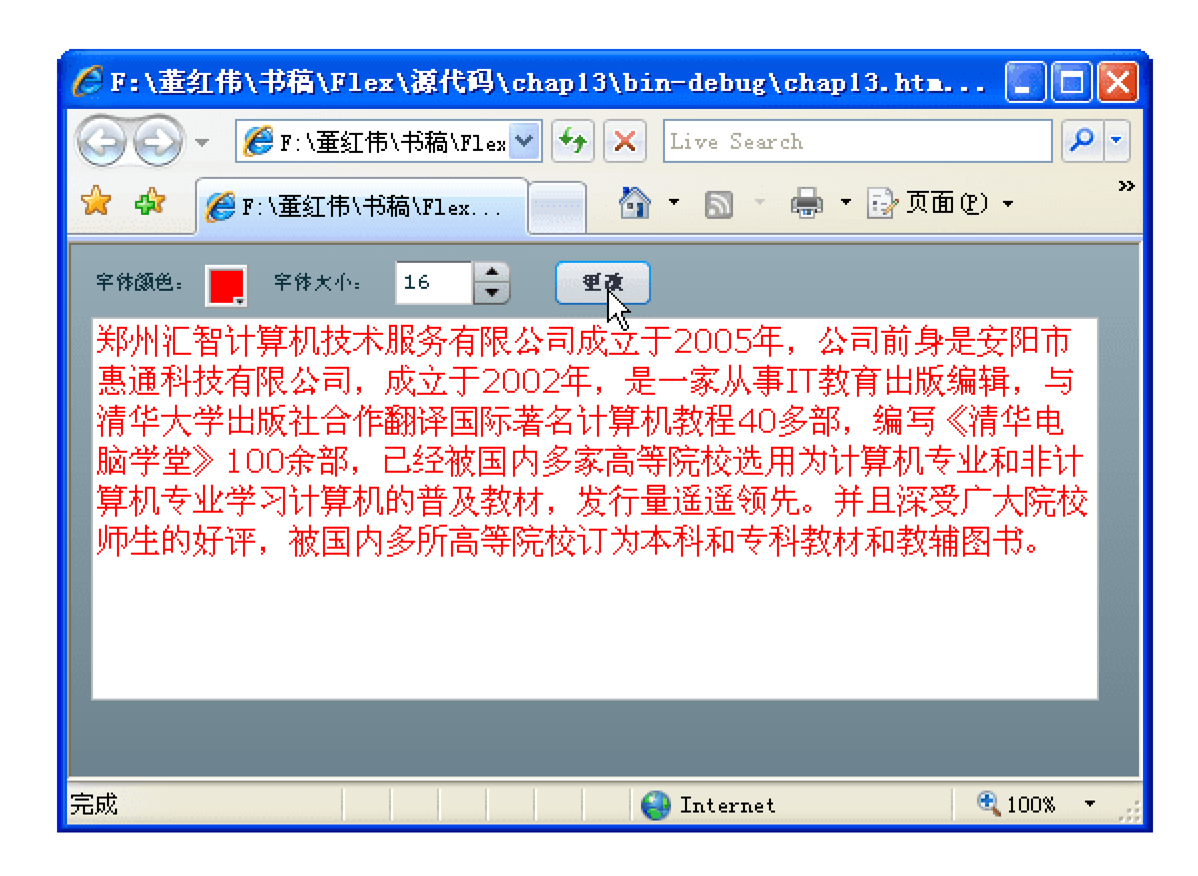


图 13-22　设置字体颜色为红色，大小为 16 时效果

13.3.4　使用主题

对 Windows 操作系统的主题，应该都不陌生，可以定制系统的操作界面。在 Flex 中，也提供了主题这个概念。其实主题就是一个配置完整的样式包，样式包定义的色彩和外观形成一种表现风格。

1. Flex 主题简介

Flex SDK 默认使用的主题名为 Halo，只包含一个 default.css 文件。该文件位于 Flex Builder 安装目录下的 sdks\3.0.0\frameworks\libs 文件夹中，用 WinZip 或 WinRAR 等解压缩工具打开目录下的 framework.swc 文件，可以看到其中的 default.css 文件。此外，还包括了 default-2.0.0.css 和 default-2.0.1.css 两个 CSS 文件，是该主题的不同版本。

将 default.css 文件从压缩文件中解压出来，使用文本编辑器打开之后，会看到里面的样式代码，定义了大部分组件的样式和外观。

主题文件一般以 SWC 文件形式存放，也可以是 CSS 样式文件和其他图片资源，推荐使用 SWC 文件形式，好处在于使用方便、利于管理。当样式由很多文件组成时，打包成一个 SWC 文件，拷贝起来会很方便，也利于反复使用。而且 SWC 文件是经过 Flex 编译器预编译的，无法修改，使用时也不用再编译，提高了运行速度。

Flex SDK 中包含了几套主题供开发者使用，它们位于 Flex Builder 的安装目录下的 sdks\3.0.0\frameworks\themes 文件夹中，该文件夹中包含的文件结构如表 13-1 所示。

表 13-1　Flex SDK 中主题文件夹文件结构

主题名	文件结构
AeonGraphical	由 AeonGraphical.css 和 AeonGraphical.swf 两个文件组成，这是 Halo 主题的图像版本
HaloClassic	haloclassic.swc 文件，Flex 早期版本的风格
Ice	Ice.css
Institutional	Institutional.css
Smoke	Smoke.css 和 smoke_bg.jpg（场景的背景图片）
Wooden	Wooden.css 和 wooden_bg.jpg（场景的背景图片）

2. 使用 Flex 主题

要在 Flex 中使用主题，需要为编译器添加参数 theme，后跟上主题文件。表明主题为用户指定的主题文件。下面将页面的主题更改为 Wooden，即木质感风格。

首先在 src 文件夹下新建一个 themes 的文件夹，将 Flex SDK 主题文件夹中 Wooden.css 和 wooden_bg.jpg 两个文件拷贝到其中。

右击导航面板中的项目名，选择 Properties（属性）项，如图 13-23 所示。弹出设置项目属性窗口，选择左栏中的 Flex Compiler 选项，设置 Flex 编译器，如图 13-24 所示。

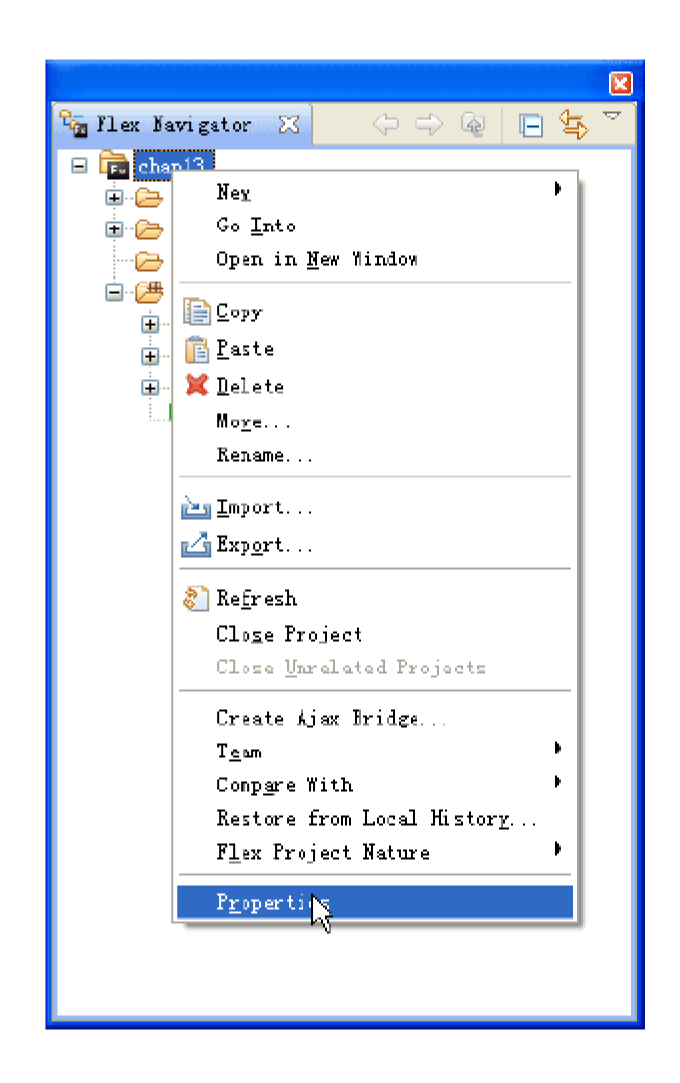


图 13-23　选择 Properties（属性）项

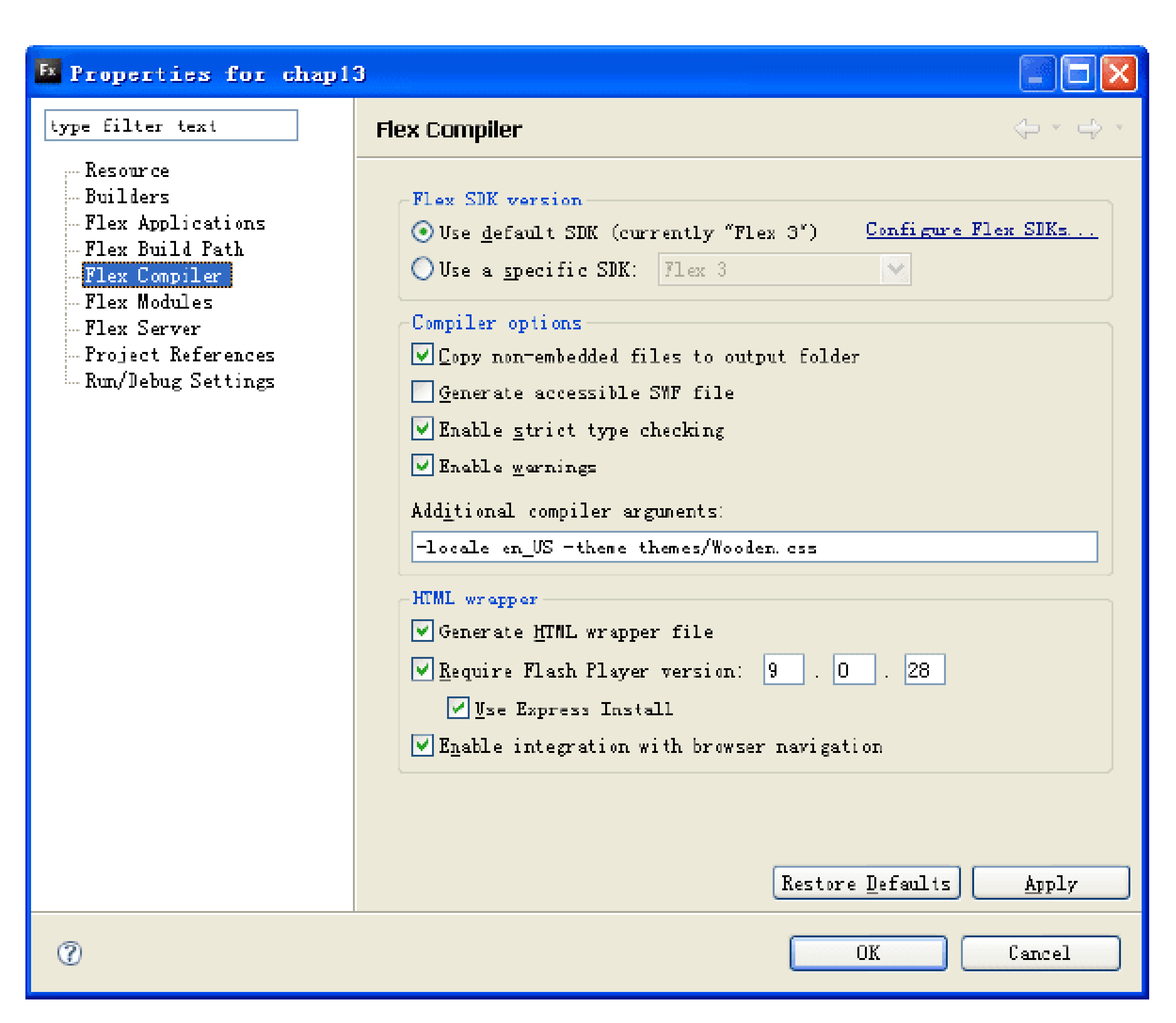


图 13-24　设置项目属性窗格

在 Additional compiler arguments 文本框中，添加-theme themes/Wooden.css 参数，单击

Apply 按钮应用主题，单击 OK 按钮完成修改。然后在程序文件中添加组件，运行后，效果如图 13-25 所示。

3. 创建 Flex 主题

通过上面的介绍，已经知道主题就是 CSS 文件和其他资源文件的组合，那么创建主题就是创建这些文件。CSS 文件的创建在第 13.3.2 小节中已经介绍过，主题 CSS 文件与一般的 CSS 文件的不同之处在于，主题 CSS 文件更复杂。

下面创建一个主题，并将该主题文件编译为 SWC 文件，然后在 Flex 项目中调用。具体步骤如下所示。



图 13-25　使用 Wooden 主题创建的页面效果

（1）在 themes 文件夹中，新建一个名为 myTheme 的 CSS 文件，在此文件中定义样式，主要是所有组件的文字大小和字体名，场景的背景图片以及 Panel 组件的一些样式。该文件的内容如代码 13.14 所示。

代码 13.14　自定义主题的 CSS 文件

```
/* CSS file */
global
{
    fontSize: 14;
    fontFamily: "宋体";
}
Application
{
    backgroundImage: Embed(source="my_bg.jpg");
}
Panel
{
    titleBackgroundSkin: Embed(source="panel_titlebg.jpg");
    backgroundImage: Embed(source="panel_bg.jpg");
    color: #FFFFFF;
}
@font-face
{
    fontFamily: "宋体";
    fontWeight: normal;
    fontStyle: normal;
    src: local("宋体");
}
```

（2）使用 Adobe Flex 3 SDK 自带的 Command Prompt 工具，该工具是以命令提示符的形式执行命令。通过输入命令，转到项目的 themes 文件夹。然后输入以下命令：

```
compc -include-file myTheme.css -include-file my_bg.jpg my_bg.jpg -include-file
panel_bg.jppg -include-file panel_titlebg.jpg panel_titlebg.jpg -o myTheme.swc
```

按回车键，执行上述命令，如果成功，将输出生成文件的绝对路径和大小，如图 13-26 所示。

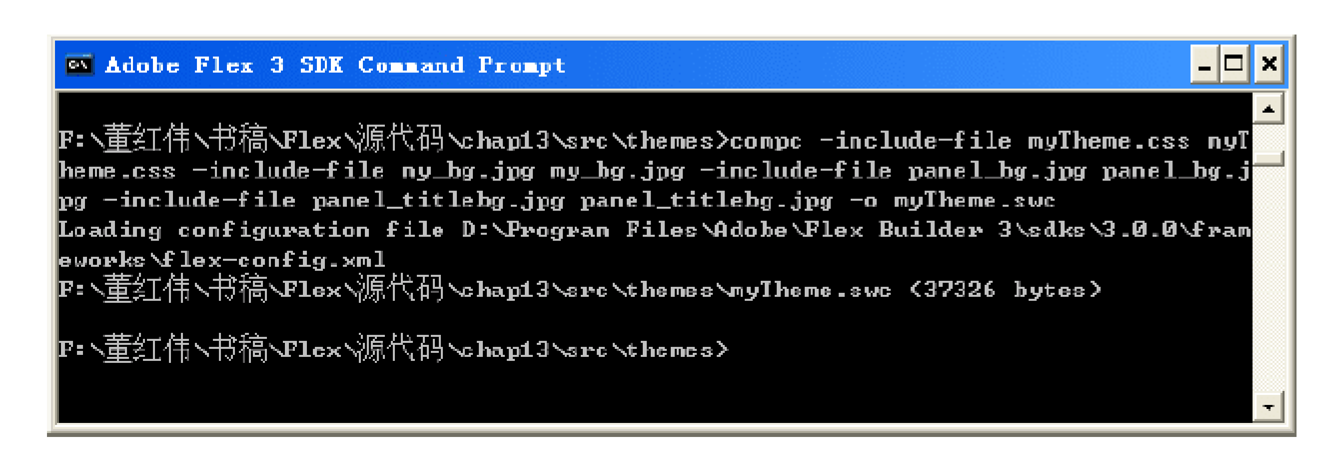


图 13-26　编译 SWC 文件成功

（3）将编译后的 myTheme.swc 文件应用到项目中。修改项目的编译器参数，如图 13-24 所示，在参数后添加“-theme themes/myTheme.swc”参数。

（4）在程序文件中，添加组件，其中要包括一个 Panel 组件，这样可以看一下定义的 Panel 组件的效果。程序文件的内容如代码 13.15 所示。

代码 13.15　创建自定义主题的程序文件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
    <mx:Button x="35" y="19" label="Button"/>
    <mx:PopUpButton x="123" y="19" label="PopUpButton"/>
    <mx:Panel x="35" y="49" width="250" height="200" layout="absolute"
    title="Panel">
    </mx:Panel>
</mx:Application>
```

（5）保存后，运行该程序文件，效果如图 13-27 所示。



图 13-27　创建自定义主题效果

13.4 参数传递

前面提到，在 Flex 中，每一个组件都是一个类。那么组件的参数传递实际上就是类对象的参数传递。类对象有属性、方法以及事件，响应的参数传递就分为属性和方法的传递，事件的传递。

13.4.1 属性的传递

可以在自定义组件中添加属性以及 get 和 set 存取器方法，实现自定义属性。在调用该组件的文件中，通过设置属性达到改变状态的效果。下面通过一个例子，演示如何使用组件属性进行传值。

（1）首先在 Components 文件夹中新建一个 MXML Component 文件，命名为 SendArg.mxml，该组件扩展自 TitleWindow 组件，大小默认。

（2）然后在新建的组件中添加 Text 组件，text 属性为“{MyText}”，绑定到 MyText 变量，该变量将在脚本中定义。该 Text 组件用来显示文本内容，大小为 100%显示。创建后的效果如图 13-28 所示。

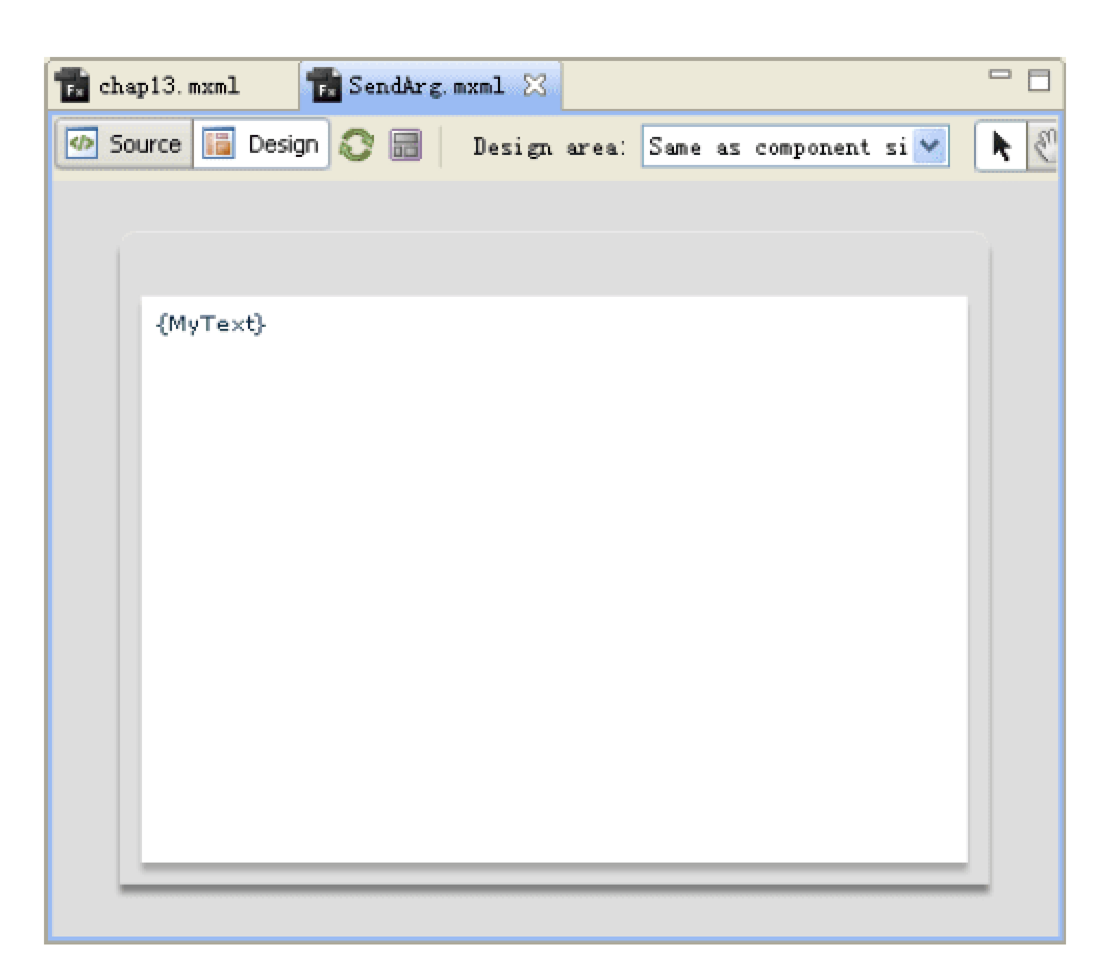


图 13-28　创建好的自定义组件效果

（3）在代码视图中，添加<mx:Script>标签，创建组件中脚本。主要是定义 MyText 属性，及该属性的 get 和 set 存取器，如代码 13.16 所示。

代码 13.16　定义组件属性和存取器

```
<mx:Script>
    <![CDATA[
        [Bindable]
        private var MyText:String="";
```

```
            public function set myText(value:String):void{
                MyText=value;
            }
            public function get myText():String{
                return MyText;
            }
        ]]>
    </mx:Script>
```

（4）在主程序中，添加供用户输入文字的 TextInput 和 TextArea 组件，分别设置 id 属性。添加创建的自定义组件，设置唯一 id 属性为 SendArg。再添加 Text 和 Button 组件，调整组件位置和大小。在 Button 组件的 click 属性中，添加 changText()函数，该函数将在脚本中定义。

（5）再在主程序文件中，添加<mx:Script>标签，创建组件中脚本，主要是定义 changText()函数。该函数将用户输入的文字，传递到自定义组件的属性中，实现状态改变。整个主程序文件的内容如代码 13.17 所示。

代码 13.17　改变自定义组件的属性

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
xmlns:ns1="Components.*" fontSize="15">
<mx:Script>
    <![CDATA[
        internal function changText():void{
            SendArg.title=tiNewTitle.text;
            SendArg.myText=taNewText.text;
        }
    ]]>
</mx:Script>
    <ns1:SendArg x="360" y="40" id="SendArg" width="318" height="207">
    </ns1:SendArg>
    <mx:Button x="124" y="259" label="修改" click="changText()"/>
    <mx:TextArea x="10" y="104" width="311" height="143" id="taNewText"/>
    <mx:TextInput x="10" y="55" width="311" id="tiNewTitle"/>
    <mx:Text x="10" y="36" text="标题文字: "/>
    <mx:Text x="10" y="83" text="内容文字: "/>
</mx:Application>
```

代码 13.17 中，语句 xmlns:ns1="Components.*"，声明 ns1 命名空间，包括 Components 包中的所有内容。<ns1:SendArg>标签，就是已创建的自定义组件。脚本中调用的自定义组件，是通过该组件在程序文件中的 id 属性调用的。

（6）保存全部文件，在浏览器中运行该程序，输入标题和内容文字，单击【修改】按钮后，可以看到自定义组件内容得到更新，如图 13-29 所示。

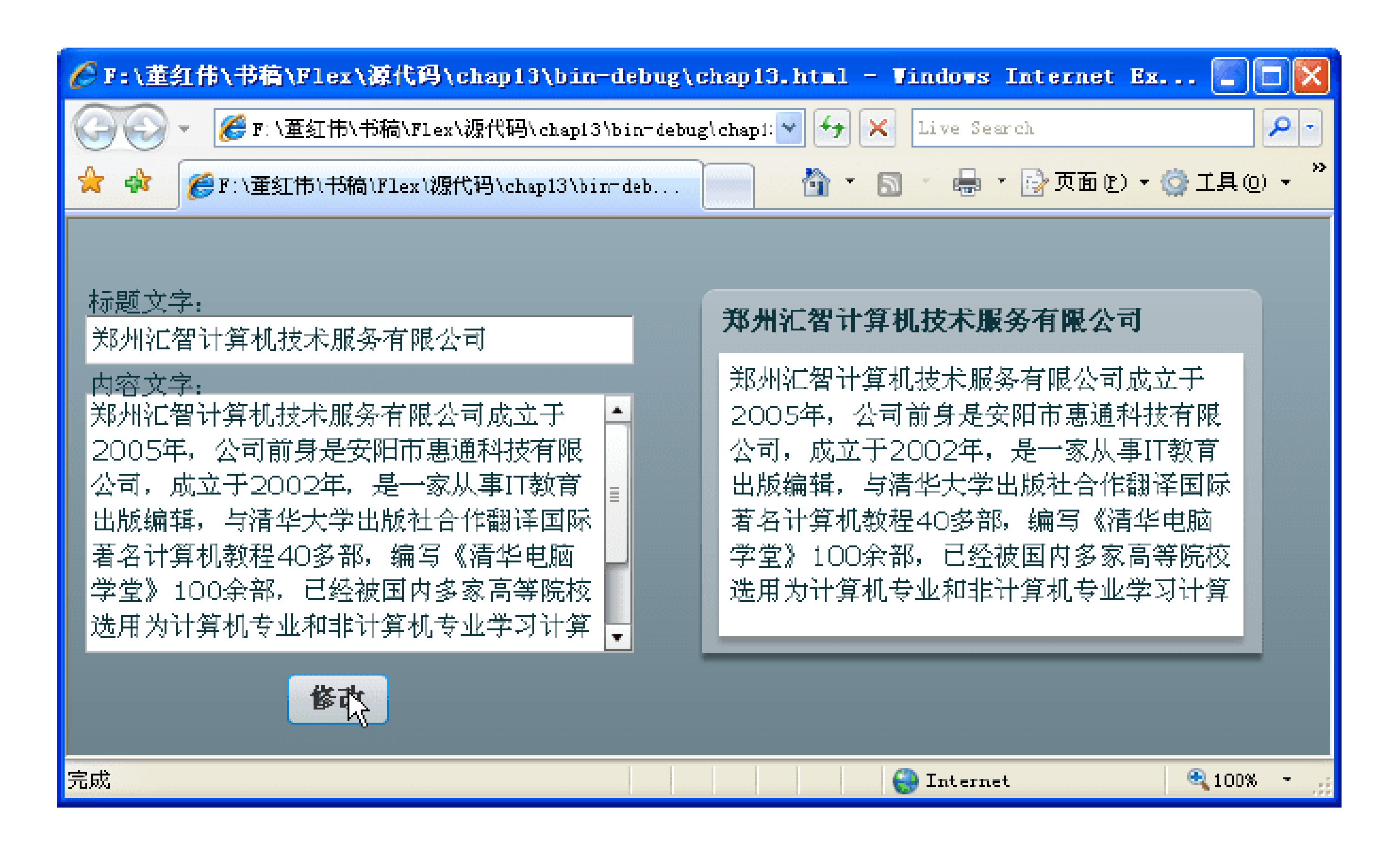


图 13-29　设置自定义组件属性

13.4.2　方法的传递

自定义组件中的方法，其传递方式与属性的传递类似。程序文件中调用自定义组件中的方法，要遵循访问规则，在第 4.1.3 小节中有介绍。

下面通过一个实例演示如何进行方法传递。该实例是一个单机版的聊天室，显示内容的部分做成用户控件。在主程序中输入信息内容，通过调用自定义组件的方法，将信息显示出来。

（1）首先在 Components 文件夹中新建一个 MXML Component 文件，命名为 SendFunction.mxml，该组件扩展自 TitleWindow 组件，大小默认。

（2）在其中添加一个 List 组件，用来显示信息内容，该组件的数据源为{arrCol}。arrCol 为一个 ArrayCollection 对象，用来保存所有信息，在脚本中创建。

（3）在自定义组件中添加<mx:Script>标签，创建脚本，包括创建 ArrayCollection 对象和 addItem()方法。addItem()方法完成将传递的参数添加到 ArrayCollection 对象。组件文件的内容如代码 13.18 所示。

代码 13.18　单机版聊天室用户控件：SendFunction.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="400" height="300" title="留言内容">
    <mx:Style>
        .mystyle {
            alternatingItemColors: #cccccc, #ffffff;
        }
    </mx:Style>
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
```

```
                [Bindable]
                protected var arrCol:ArrayCollection=new ArrayCollection;
                public function addItem(ItemText:Object):void{
                    var NewItem:Object=ItemText;
                    arrCol.addItem(NewItem);
                }
            ]]>
        </mx:Script>
        <mx:List x="0" y="0" id="li1" width="100%" height="100%" dataProvider=
        "{arrCol}" styleName="mystyle"></mx:List>
    </mx:TitleWindow>
```

（4）在主程序文件中添加自定义组件，设置 id 属性为 mySendFunction。添加 Text、TextInput 和 Button 组件，设置相关属性，为用户提供输入功能，主程序文件内容如代码 13.19 所示。

代码 13.19　单机版聊天室主程序内容

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
fontSize="15" xmlns:ns1="Components.*">
    <ns1:SendFunction x="38.5" y="10" id="mySendFunction">
    </ns1:SendFunction>
    <mx:TextInput x="83" y="328" width="279" id="tiNewText"/>
    <mx:Text x="38" y="330" text="留言: "/>
    <mx:Button x="370" y="328" label="提交" click="SendText()"/>
</mx:Application>
```

（5）在主程序中添加<mx:Script>标签，创建脚本，主要包括 SendText()方法的定义。该方法是页面中按钮对象的单击事件处理函数，实现调用自定义组件的 addItem()方法，如代码 13.20 所示。

代码 13.20　主程序中调用自定义组件方法

```
<mx:Script>
    <![CDATA[
        internal function SendText():void{
            mySendFunction.addItem(this.tiNewText.text);
        }
    ]]>
</mx:Script>
```

（6）保存全部文件，在浏览器中运行，输入信息，单击【提交】按钮，会看到自己的信息显示在自定义组件中，如图 13-30 所示。

13.4.3　事件的传递

在第 12 章已学习了事件相关知识，在 Flex 中，自定义组件的事件传递较为复杂，特别是

用到自定义事件。由于在开发程序中，仅仅依靠系统提供的事件处理机制，有很多效果实现起来都非常困难。

用户可以扩展系统提供的事件，创建符合自己要求的事件。在 Flex 中，创建自定义事件类，由此类管理程序中的事件，这是一种好的编程习惯。这样做可以降低程序的耦合度，使程序结构更加清晰。

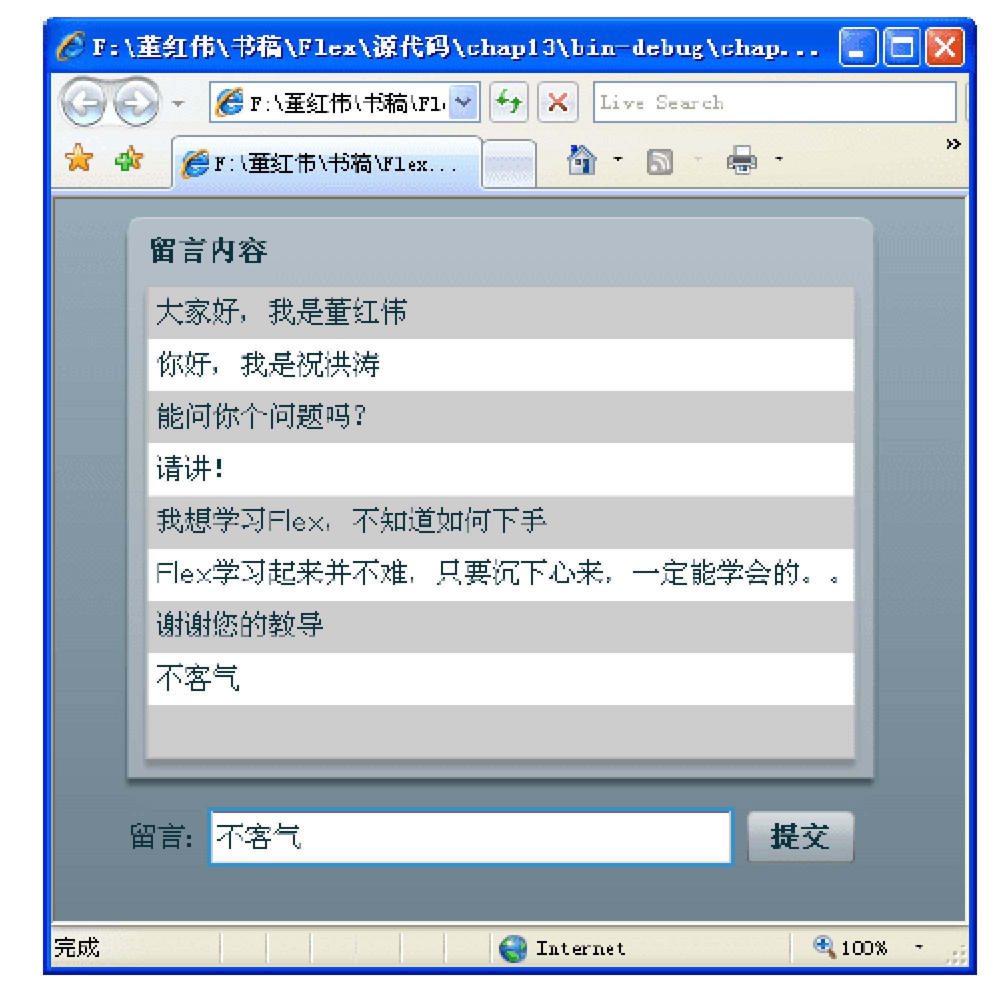


图 13-30　单机版聊天室效果

下面通过一个简单的购物车程序，演示如何创建自定义事件类，以及如何进行事件的传递。该购物车采用 DataGrid 组件显示产品，CheckBox 组件实现选择，List 组件用来显示选中的产品。

（1）首先在项目的 src 目录下创建名为 bookEvent 文件夹，该文件夹用来存放事件处理类。

（2）在 bookEvent 文件夹中创建一个名为 cartEvent 的类文件，该类继承自 Event 类。在该类中添加两个属性：isAdd 和 book，分别代表是否选中产品和该产品对应的信息集合对象。然后定义构造函数，该函数为两个属性定义新值，并调用超类的构造函数，传递 AddBook 参数。该文件的内容如代码 13.21 所示。

代码 13.21　自定义 cartEvent 类：cartEvent.as

```
package bookEvent{
    import flash.events.Event;
    public class cartEvent extends Event{
        public var isAdd:Boolean;
        public var book:Object;
        function cartEvent(_data:Object,_isAdd:Boolean):void{
            isAdd=_isAdd;
            book=_data;
            super("AddBook");
        }
    }
}
```

（3）然后在 Components 文件夹中，新建一个 MXML Component 文件，命名为 SendEvent.mxml，该组件扩展自 Canvas 组件，大小默认。

（4）在新建的自定义组件中添加一个 CheckBox 组件，设置 click 属性为 addCart()函数。addCart()函数实例化自定义事件类，并由主程序发送该事件对象到事件流。SendEvent.mxml 文件的内容如代码 13.22 所示。

代码 13.22　创建自定义组件：SendEvent.mxml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Style>
        .myCheckBox{
            color:#ff0000;
        }
    </mx:Style>
    <mx:Script>
        <![CDATA[
            import bookEvent.cartEvent;
            internal function addCart():void{
                this.parentApplication.dispatchEvent(new cartEvent(data,
                cart_check.selected));
            }
        ]]>
    </mx:Script>
    <mx:CheckBox x="0" y="0" label="购买" id="cart_check" click="addCart()"
    styleName="myCheckBox"/>
</mx:Canvas>
```

（5）在主程序中添加一个<mx:Model>标签，并添加内容，作为产品的数据源，如代码 13.23 所示。

代码 13.23　添加数据源

```
<mx:Model id="books">
    <datas>
        <book>
            <id>0</id>
            <name>Flash 第 1 步</name>
            <author>董红伟</author>
            <date>2006 年</date>
        </book>
        <book>
            <id>1</id>
            <name>Flash 第 2 步</name>
            <author>董红伟</author>
            <date>2007 年</date>
        </book>
        <book>
            <id>2</id>
            <name>Flash 第 3 步</name>
            <author>董红伟</author>
            <date>2008 年</date>
        </book>
    </datas>
</mx:Model>
```

（6）在主程序中添加一个 DataGrid 组件，设置该组件的数据源为代码 13.23 中定义的数据源，并设置 DataGridColumn 模板，如代码 13.24 所示。

代码 13.24　添加 DataGrid 组件

```
<mx:DataGrid x="10" y="10" dataProvider="{books.book}" width="390">
    <mx:columns>
        <mx:DataGridColumn headerText="书名" dataField="name"/>
        <mx:DataGridColumn headerText="作者" dataField="author"/>
        <mx:DataGridColumn headerText="出版日期" dataField="date"/>
        <mx:DataGridColumn headerText="购买" itemRenderer="Components.
        SendEvent"/>
    </mx:columns>
</mx:DataGrid>
```

（7）再添加一个 ArrayCollection 对象，用来保存选中的产品，添加一个 Text 和 List 组件，用来显示选中的书籍信息，如代码 13.25 所示。

代码 13.25　添加显示选中产品的相关组件

```
<mx:ArrayCollection id="arrCol"/>
<mx:Text x="10" y="160" text="您购买的书籍如下所示: "/>
<mx:List x="10" y="192" width="390" dataProvider="{arrCol}" labelField="name">
</mx:List>
```

（8）添加<mx:Script>标签，在其中定义一个 init()函数和一个 AddBookHandler()函数，如代码 13.26 所示。

代码 13.26　定义添加事件侦听器和事件处理的函数

```
<mx:Script>
    <![CDATA[
        import bookEvent.cartEvent;
        internal function init():void{
            addEventListener("AddBook",AddBookHandler);
        }
        internal function AddBookHandler(event:cartEvent):void{
            if(event.isAdd){
                arrCol.addItem(event.book);
            }else{
                arrCol.removeItemAt(arrCol.getItemIndex(event.book));
            }
        }
    ]]>
</mx:Script>
```

init()函数为程序添加一个事件侦听器，该侦听器侦听类型为 AddBook 的事件，当侦听到时，由 AddBookHandler()事件处理函数进行处理。

AddBookHandler()事件处理函数，参数为 cartEvent 对象，判断该对象的 isAdd 值。如果为真，将该对象的 book 成员添加到 ArrayCollection 对象中，反之，从 ArrayCollection 对象中删除。

（9）在程序的<mx:Application>标签内设置 creationComplete 事件处理函数，为 init()函数，也就是场景创建完成之后执行 init()函数，如代码 13.27 所示。

代码 13.27　为主程序添加初始函数

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
fontSize="15" xmlns:ns1="Components.*" creationComplete="init()">
…
</mx:Application>
```

（10）保存所有文件，在浏览器中运行，启用【购买】复选框，显示出选中的项，效果如图 13-31 所示。

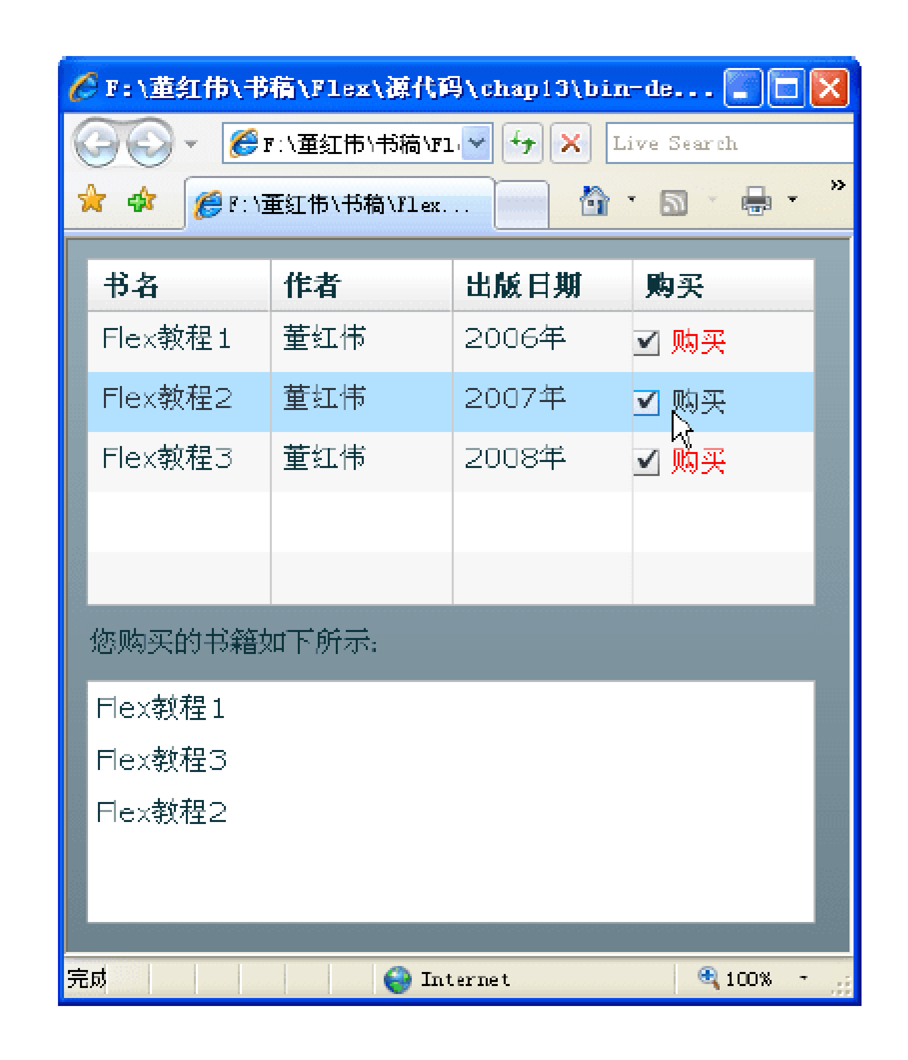
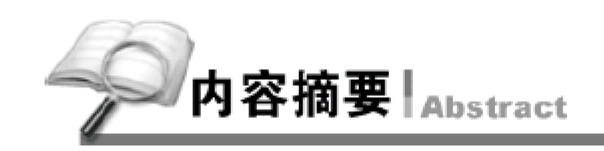


图 13-31　查看购买列表

第 4 篇　Flex 数据交互篇

第14章 Flex 中的数据处理



数据是程序处理的主要对象。在应用程序中最核心的部分也是围绕对数据的处理展开的。本章将介绍如何在 Flex 3.0 中对数据进行处理，包括数据模型、数据绑定、数据显示组件 DataGrid、数据验证和数据格式化等内容。



- 掌握在 MXML 文件中定义数据模型的方法
- 掌握数据在 Flex 中的作用及应用
- 掌握使用“{}”和“[Bindable]”的数据绑定方法
- 熟悉<mx:Binding>组件的使用
- 掌握 DataGrid 组件显示和编辑数据的方法
- 掌握如何对 Flex 中的各种数据进行验证
- 掌握数据格式化的使用方法

14.1 数据模型

一个数据模型就是一个 ActionScript 对象，这个对象的属性用来存储应用程序所需的数据。在向服务器发送数据之前，或者从服务器接收数据但还没有使用之前，数据模型提供一个在 Flex 应用程序中存储数据的途径。Adobe Flex 应用程序与服务器之间的通信只发生在 Flex 应用程序需要检索的数据不可用时和使用新数据更新服务器端的数据源时。

另外，在 Flex 中使用的数据模型除了可以到服务器获取，还可以使用 MXML 或 ActionScript 静态对象，或者从本地 XML 文件中获取。

14.1.1 使用<mx:Model>组件

使用<mx:Model>组件是基于 MXML 标记定义数据的最通用方式，这个组件会被编译为 ActionScript 对象。当数据具有层次关系时会被编译为一系列树状对象，这些编译后的对象是没有类型信息的。对象树的叶子是可固定的值。因为模型定义在<mx:Model>组件中时不包含类型信息或业务逻辑，它们只适用于简单的需求。如果要定义属性类型或者要添加业务逻辑，

则可以在 ActionScript 中定义数据模型。

既可以在 MXML 文件中使用<mx:Model>组件定义数据，又可以使用组件的 source 属性从额外的数据文件中加载数据。当使用 source 属性时，额外的数据文件被编译进 SWF 文件，而不是在运行时加载。

嵌入在组件或者外部文件中的数据模型声明，必须由单独的根节点包含所有子节点。用户可以使用 MXML 绑定表达式，例如在模型中声明{CardName.text}。这个方式还可以绑定 form 字段的内容到一个结构化的数据描述。

<mx:Model>组件存储数据的语法如下所示。

```
<mx:Model id="Modelid">
   <根节点>
      <节点 1/>
      <节点 2/>
      …
   </根节点>
</mx:Model>
```

<mx:Model>组件定义数据时必须有根节点，即<mx:Model>标签下有一节点能包括全部的其他节点。

例如，在下面的例子中，会使用<mx:model>定义一个数据模型来保存联系人的数据。这个数据模型使用数据绑定来接收数据界面中各种 form 控件的数据。因此，无论输入怎么变化，Name 节点从 TextInput 组件 CardName 的 text 属性中获得数据。类似的，modelCard 数据模型中的 ID 和 E-mail 属性值使用数据绑定从 CardID 和 CardEmail 文本域中获得。

首先新建一个名为 chap14 的 Flex Project，本章中的所有程序都在 chap14 下进行。然后，添加一个新的 MXML Application 命名为 ModelData.mxml。

在 ModelData 中添加一个 Panel 组件和 Form 表单，向表单中添加用户可输入的联系人信息项，这里包括编号、姓名、邮箱、职业和网站 5 项，如代码 14.1 所示。

代码 14.1　添加联系人 Panel

```
<mx:Panel width="300" height="252" title="添加新联系人" horizontalAlign=
"center">
    <mx:Form >
       <mx:FormItem label="编号: ">
          <mx:TextInput id="CardID"/>
      </mx:FormItem>
       <mx:FormItem label="姓名: ">
          <mx:TextInput id="CardName"/>
      </mx:FormItem>
       <mx:FormItem label="邮箱: ">
          <mx:TextInput id="CardEmail"/>
      </mx:FormItem>
       <mx:FormItem label="职业: ">
          <mx:TextInput id="CardCareer"/>
```

```
          </mx:FormItem>
           <mx:FormItem label="网站: ">
              <mx:TextInput id="CardSite"/>
          </mx:FormItem>
      </mx:Form>
     <mx:Button label="确定"  click="AddNewCard(event)" />
</mx:Panel>
```

添加之后，再为应用程序设置一个背景色，然后返回到 Design 模式中可看到类似图 14-1 所示的效果，显示了 Panel 组件及嵌入的添加联系人表单。

接下来，使用<mx:Model>组件来定义一个联系人数据模型，包括上述需要输入的 5 项，如代码 14.2 所示。

代码 14.2　<mx:Model>组件定义联系人数据模型

```
<mx:Model id="modelCard">
<User>
    <ID>{CardID.text}</ID>
    <Name>{CardName.text}</Name>
    <Email>{CardEmail.text}</Email>
    <Career>{CardCareer.text}</Career>
    <Site>{CardSite.text}</Site>
</User>
</mx:Model>
```

这里为<mx:Model>组件数据模型定义了 id 属性为 modelCard，数据被包含在</User>根节点，每个子节点对应一个表单输入项。当使用 modelCard 指向<mx:Model>数据模型时，会跳过根节点，所以这里要获取 Site 节点的值，可以使用 modelCard.Site。

添加如代码 14.3 所示的 ActionScript，再保存并按 Ctrl+F11 组合键运行程序。在各个 TextInput 组件中依次输入联系人的信息，再单击【确定】按钮，结果将弹出对话框显示，如图 14-2 所示。

代码 14.3　显示结果

```
<mx:Script>
    <![CDATA[
        import mx.controls.Alert;
        internal function AddNewCard(e:MouseEvent):void
        {
            var msg:String;
            msg="编号: "+modelCard.ID+"\r";
            msg+="姓名: "+modelCard.Name+"\r";
            msg+="邮箱: "+modelCard.Email+"\r";
            msg+="职业: "+modelCard.Career+"\r";
            msg+="网站: "+modelCard.Site;
            Alert.show(msg,"添加成功");
        }
```

```
    ]]>
</mx:Script>
```



图 14-1　Panel 组件效果

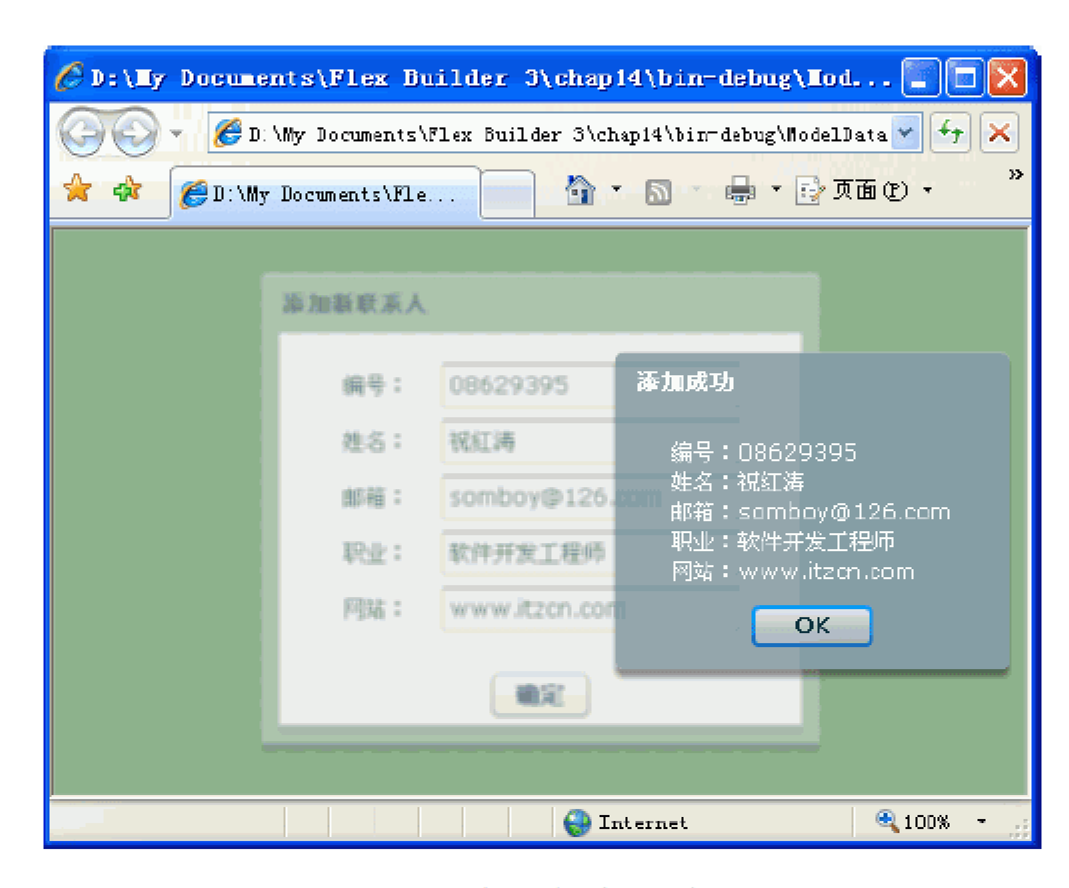
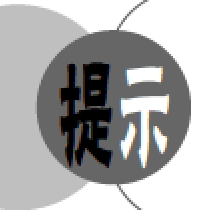


图 14-2　添加联系人效果

14.1.2　使用<mx:XML>组件

Flex 3.0 允许用户使用 XML 定义数据模型，因为 Flex 3.0 在新的 ECMAScript for XML（E4X）标准中对 XML 的处理能力有了很大提高。E4X 定义了新的类与功能集合，使得处理 XML 更加简单。



可以使用<mx:XML>或者 ActionScript 创建基于 XML 的数据模型。在 ActionScript 3 中，XML 是一种本地数据类型。

<mx:XML>组件用于在 Flex 应用程序中定义 XML 数据。<mx:XML>组件的定义语法与<mx:Model>组件定义语法相近，都为树型数据定义。其定义语法如下所示。

```
<mx:XML id="XML 组件 id" format="e4x">
    <根节点>
        <节点 1/>
        <节点 2/>
        …
    </根节点>
</mx:XML>
```

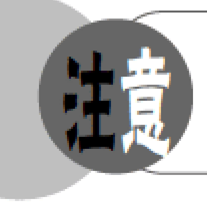
format 属性设置为 e4x，可指定创建的 XML 对象使用 E4X 标准实现。为了向后兼容，在没有明确指定 format 属性为 e4x 时，对象的类型被设置为 flash.xmlXMLNode。

例如，代码 14.4 使用<mx:XML>组件实现了代码 14.2 的实例中<mx:Model>组件定义的数据模型。

代码 14.4　<mx:XML>组件定义联系人数据模型

```
<mx:XML id="modelCard" format="e4x">
```

```
<User>
    <ID>{CardID.text}</ID>
    <Name>{CardName.text}</Name>
    <Email>{CardEmail.text}</Email>
    <Career>{CardCareer.text}</Career>
    <Site>{CardSite.text}</Site>
</User>
</mx:XML>
```



读者可对代码 14.2 和 14.4 进行比较，并注意两者的相似处。

14.1.3 使用<mx:Object>组件

<mx:Object>组件是一种抽象的数据模型组件，可用于定义复杂数据。使用<mx:Object>组件存储数据的语法如下所示。

```
<mx:Object 属性名 1="属性值 1" 属性名 2="属性值 2" … 属性名 n="属性值 n"/>
```

<mx:Object>组件使用属性来存储数据。一个<mx:Object>组件可看成一行数据，多个<mx:Object>组件就组成了类似表格的复杂数据模型。例如，代码 14.5 使用<mx:Object>组件定义联系人数据模型，并在<mx:DataGrid>组件中显示。

代码 14.5 <mx:Object>组件定义联系人数据模型

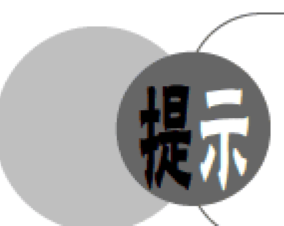
```
<mx:Panel width="546" height="219"  title="查看联系人列表"  horizontalAlign=
"center">
  <mx:DataGrid width="100%">
     <mx:columns>
       <mx:DataGridColumn headerText="编号" dataField="ID" width="60"/>
       <mx:DataGridColumn headerText="姓名" dataField="Name"/>
       <mx:DataGridColumn headerText="邮件" dataField="Email"/>
       <mx:DataGridColumn headerText="职业" dataField="Career"/>
       <mx:DataGridColumn headerText="网站" dataField="Site"/>
    </mx:columns>
    <mx:dataProvider>
    <mx:Object ID="E5091" Name="王梦呓" Email="ayhncn@163.com" Career="销售
    主管" Site="www.itzcn.net"/>
    <mx:Object ID="I9395" Name="祝红涛" Email="somboy@126.com" Career="软件
    开发工程师" Site="www.itzcn.com "/>
    <mx:Object ID="C0892" Name="贺强" Email="Qiange@188.com" Career="市场策
    划员" Site="www.webzcn.cn"/>
    </mx:dataProvider>
   </mx:DataGrid>
</mx:Panel>
```

将代码 14.5 添加到一个 MXML Application 文件中，程序的运行效果如图 14-3 所示。

图 14-3 <mx:Object>组件定义数据模型效果

14.1.4 使用 ActionScript 脚本

作为可替代基于 MXML 模型的方式，Flex 3.0 还允许用户在<mx:Script>标记中使用 ActionScript 脚本定义数据模型。下面仍然以添加联系人信息为实例来介绍这种方式，不同的是在 ActionScript 脚本中定义所需的联系人数据模型。



使用大括号"{}"语法的数据绑定方式仅适用在 MXML 标记中，如 14.1.1 节和 14.1.2 节的示例。在 14.2 节会对数据绑定进行详细介绍。

首先新建一个名为 ASData.mxml 的 Flex 程序，为根节点 Application 添加“creation-Complete="initApp()"”，再添加<mx:Script>标记。在 ActionScript 脚本块中使用代码 14.6 定义联系人数据模型。

代码 14.6　ActionScript 定义联系人数据模型

```
<mx:Script>
    <![CDATA[
        private var modelCard:Object={ID:"",Name:"",Email:"",Career:"",
        Site:"" }
    ]]>
</mx:Script>
```

在 initApp()事件处理器中，增加 UpdateModelData()方法作为 form 元素的 change 事件处理器，如代码 14.7 所示。

代码 14.7　initApp 事件处理程序

```
internal function initApp():void
{
```

```
CardID.addEventListener(Event.CHANGE, UpdateModelData);
CardName.addEventListener(Event.CHANGE, UpdateModelData);
CardEmail.addEventListener(Event.CHANGE, UpdateModelData);
CardCareer.addEventListener(Event.CHANGE, UpdateModelData);
CardSite.addEventListener(Event.CHANGE, UpdateModelData);
}
```


UpdateModelData()方法实现前面使用“{}”完成的功能，即获取用户的输入并更新到数据模型中，如代码 14.8 所示。

代码 14.8　UpdateModelData()方法

```
internal function UpdateModelData(e:Event):void
{
    modelCard.ID=CardID.text,            //编号
    modelCard.Name=CardName.text,        //姓名
    modelCard.Email=CardEmail.text,      //邮箱
    modelCard.Career=CardCareer.text,    //职业
    modelCard.Site=CardSite.text         //网站
}
```

最后添加 Panel 组件和 Form 表单，这与前面相同，这里不再重复。同时，单击【确定】按钮执行的 AddNewCard()方法也相同。运行程序，ASData.mxml 执行效果如图 14-4 所示。

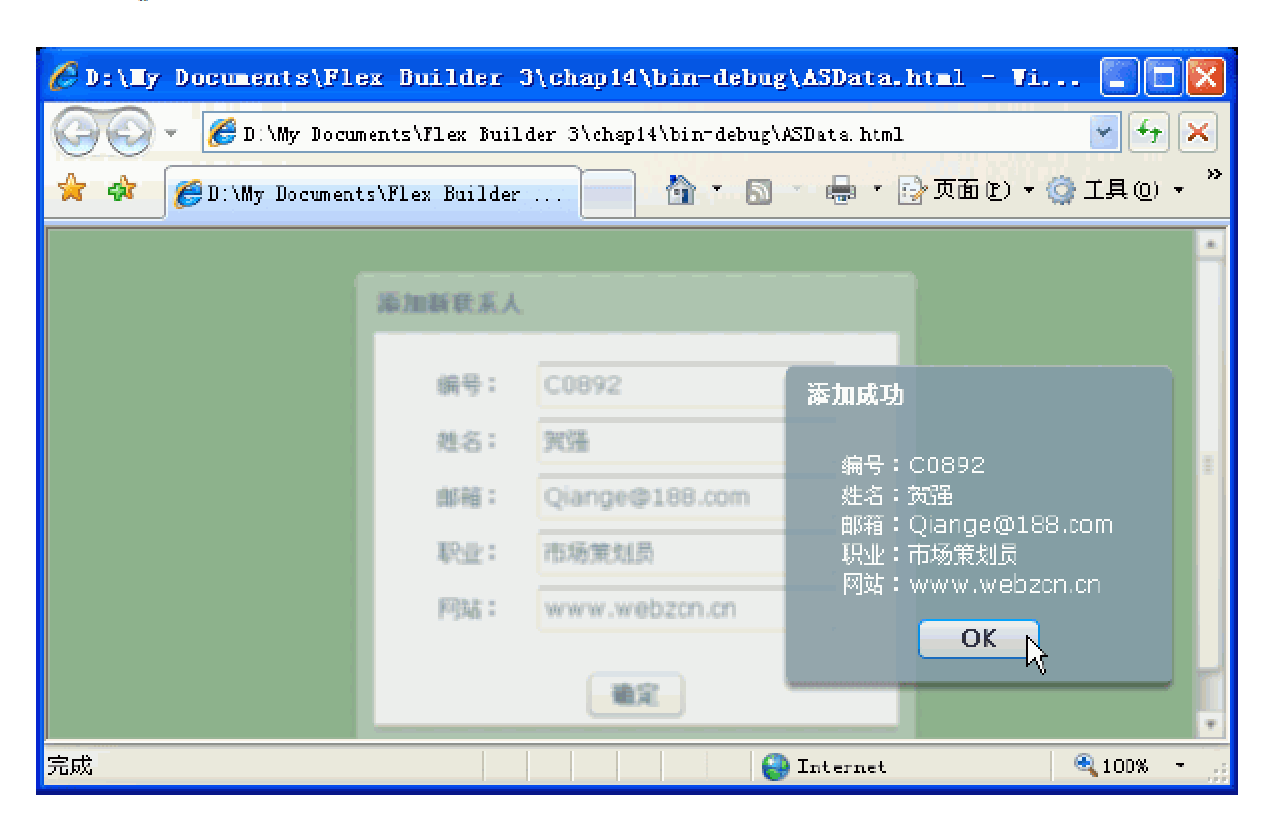


图 14-4　使用 ActionScript 脚本效果

14.1.5　使用类

使用基于 ActioinScript 脚本，或基于 MXML 的数据模型时，用户无法自定义属性类型。要定义属性类型，必须使用基于类的数据模型。

当使用带类型的属性存储复杂数据结构，或者想要应用程序数据执行客户端业务逻辑时，使用类作为数据模型是一个非常好的选择。当数据模型传递到服务器端数据服务时，基于类的数据模型的类型信息也会被保留下来。

代码 14.9 所示为联系人数据模型被定义为 CardModel 的 ActionScript 类。CardModel 包含各种类型的属性，还有存储方法，简单的自定义校验方法 ValidateModel()和一个返回类成员信息的方法 GetAll()。

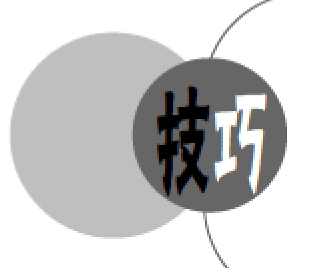
代码 14.9　定义联系人数据模型的 CardModel 类

```
package
{
    public class CardModel
    {
        //声明用于检验的变量
        [Bindable]
        public var isValid:Boolean=false;

        //声明联系人数据成员
        private var _ID:String="";
        private var _Name:String="";
        private var _Email:String="";
        private var _Career:String="";
        private var _Site:String="";
        //存储编号方法
        public function set ID(Value:String):void{
            _ID=Value;
            ValidateModel();
        }
        //存储姓名方法
        public function set Name(Value:String):void{
            _Name=Value;
            ValidateModel();
        }
        //存储邮箱方法
        public function set Email(Value:String):void{
            _Email=Value;
            ValidateModel();
        }
        //存储职业方法
        public function set Career(Value:String):void{
             Career=Value;
            ValidateModel();
        }
        //存储网站方法
        public function set Site(Value:String):void{
            _Site=Value;
            ValidateModel();
        }
        //有效性校验方法
        private function ValidateModel():void{
```

```
                isValid=_ID!=""&&_Name!=""&&_Email!=""&&_Career!=""&&_Site!="";
            }
            //获取成员信息方法
            public function GetAll():String{
                var msg:String;
                msg="编号: "+_ID+"\r";
                msg+="姓名: "+_Name+"\r";
                msg+="邮箱: "+_Email+"\r";
                msg+="职业: "+_Career+"\r";
                msg+="网站: "+_Site;
                return msg;
            }
        }
    }
```

创建一个名为 CardModel.as 的类文件并添加代码 14.9。ValidateModel()方法仅包含了最简单的业务逻辑，判断是否为空值并保存到 isValid 变量中。GetAll()方法包含了描述联系人属性的各种信息，最终以字符串返回。



GetAll()方法在使用不同技术的层次间传递数据时特别有用。例如 Flash Remoting（一种数据传输机制，Flex Data Service 也使用这种机制）和 JSON，它是序列化与反序列化对象，用于存取不同对象的类型信息。

当使用一个基于类的数据模型时，可以在 ActionScript 脚本或者 MXML 中实例化它们。在下面的例子中，使用 ActionScript 脚本实例化数据模型，并且设置它的名称为 modelCardData。然后，在 ActionScript 脚本中调用属性绑定到 Form 组件的数据模型上，并且使用 Change 事件来检测 Form 组件的值并更新到数据模型中。具体步骤如下。

新建一个名为 ClassData.mxml 的 MXML Application 文件，并按照前面的步骤添加 Panel 组件和 Form 输入表单，其中还包括 initApp()方法的代码。

然后，使用 ActionScript 脚本声明一个变量，其类型为上面 ActionScript 类定义的 CardModel，如代码 14.10 所示。

代码 14.10　声明 CardModel 类型变量

```
[Bindable]
private var modelCardData:CardModel=new CardModel();   //声明
```

initApp()方法监听 change 事件并交由 UpdateModelData()处理器处理，UpdateModelData()将用户的输入更新到数据模型中，其代码如代码 14.11 所示。

代码 14.11　UpdateModelData()处理器

```
internal function UpdateModelData(e:Event):void
{
```

```
    modelCardData.ID=CardID.text;
    modelCardData.Name=CardName.text;
    modelCardData.Email=CardEmail.text;
    modelCardData.Career=CardCareer.text;
    modelCardData.Site=CardSite.text;
}
```

在 CardModel 类中定义了模型的校验方法 ValidateModel()，方法将检验结果反映到 isValid 变量。在 MXML 中对【确定】按钮进行修改，添加 enabled 属性。该属性是一个布尔值用于表示是否可用，代码 14.12 所示是修改后的【确定】按钮。

代码 14.12 修改【确定】按钮

```
<mx:Button label="确定" enabled="{modelCardData.isValid}" click=
"AddNewCard(event)" />
```

代码 14.12 中 click 单击事件处理器 AddNewCard()将结果直接弹出显示，如代码 14.13 所示。

代码 14.13 AddNewCard()

```
internal function AddNewCard(e:MouseEvent):void
{
    var msg:String;
    msg=modelCardData.GetAll();
    Alert.show(msg,"添加成功");
}
```

如代码 14.13 所示，调用 CardModel 类的 GetAll()方法将数据模型中成员信息返回，再显示。运行程序，在联系人表单中输入信息，此时【确定】按钮的状态为不可用，直到输入完所有项，如图 14-5 所示。最后，在输入完成后单击该按钮查看结果，如图 14-6 所示。

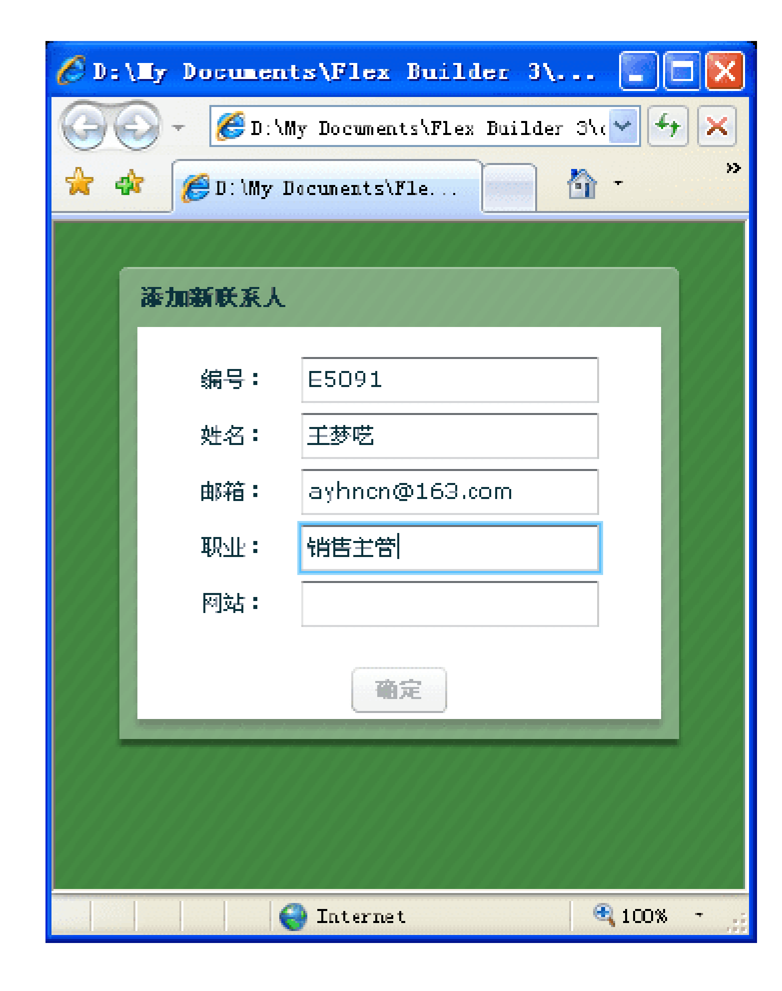


图 14-5 输入不完整时按钮不可用

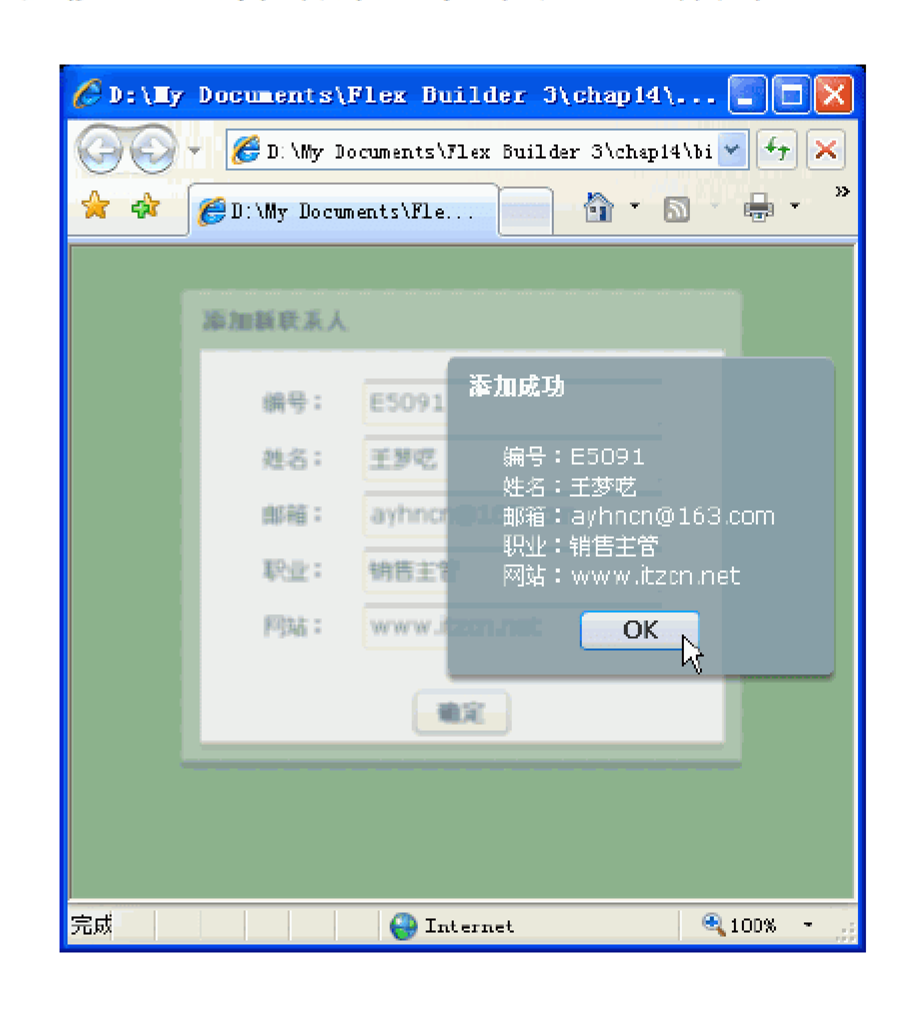


图 14-6 输入完成后查看结果

14.2　数据绑定

数据绑定是指用户可以将某个属性与其他的属性或变量相关联。这样，当被绑定的属性或变量发生改变时，绑定项同时发生改变。数据绑定是 Flex 中一项非常有用的技术，可以有效地减少程序代码、提高运行效率。Flex 提供了很多数据绑定方法，本节将把组件实例与各种数据源绑定，也可将数据绑定到 ActionScript 对象。例如，可以将数据绑定到 DataGrid 组件或者 ArrayCollection 集合中。

14.2.1　简单绑定方式

Flex 3.0 中绑定数据最简单的方式是使用大括号运算符“{}”，其语法如下：

```
{可绑定的属性、变量或者表达式}
```

大括号中的绑定表达式可以被包含在 ActionScript 表达式中用来发回一个结果。例如，用户能够在大括号运算符中绑定如下类型的内容。

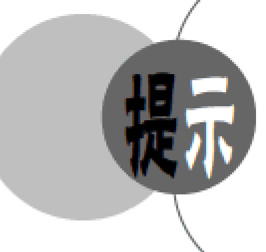
- 绑定一个单独属性。
- 使用字符串连接符，其中包括可绑定的属性。
- 对可绑定属性的各种运算。
- 使用条件运算来判断一个可绑定属性的值。

另外，Flex 3.0 还可以通过使用[Bindable]关键字的方式指定变量或属性为可绑定，其语法如下所示。

```
[Bindable]
变量或属性
```

例如，下面代码将 configSite 定义为可绑定变量，并赋予字符串值“www.itzcn.com”。

```
[Bindable]
private var configSite:String="www.itzcn.com";
```



Flex 中系统组件的大部分属性都是可绑定的，因为在组件定义中属性一般都指明为[Bindable]。对于要绑定的变量需要先声明为[Bindable]。若用户使用“{}”操作符绑定未声明为可绑定的变量时，编译会提示警告，但运行正常。

下面创建一个实例，讲解这两种方式的使用。例子演示了一个 Label 组件和一个 Box 组件绑定获得 Hslider 控件的属性值。在大括号中的属性名是绑定的源属性。当源属性的值发生变化时，Flex 复制源属性的当前值 slider.value 到目标属性，即 Box 组件的 backgroundColor 属性。Label 组件的 text 属性则使用大括号绑定表达式，在表达式中有一个使用[Bindable]关键字声明的可绑定变量。步骤如下所示。

（1）首先新建一个名称为 SimpleBind.mxml 的 MXML Application 文件。

（2）添加一个 Panel 组件，设置 title 属性为“数据的简单绑定”、horizontalAlign 属性为 center，并调整其宽度和高度。

（3）在 Panel 内添加一个 VBox 组件，再到 VBox 组件内依次添加两个 Label 组件、一个 HSlider 组件和一个 Box 组件。

（4）对添加的组件进行设置，并在属性中添加绑定。代码 14.14 所示为完成后 Panel 组件的代码。

代码 14.14　添加绑定后 Panel 组件的代码

```
<mx:Panel width="320" height="252"  title="数据的简单绑定"  horizontalAlign=
"center">
  <mx:VBox>
     <mx:Label text="拖动滑块来改变颜色"/>
     <mx:HSlider width="{box.width}" id="slider" minimum="0x000000"
     maximum="0xFFFFFF"
      liveDragging="true"  dataTipFormatFunction="intToHex" />
     <mx:Box id="box" width="200" height="100" backgroundColor="{slider.
     value}" />
 <mx:Label text="{'当前值:'+strColor}" />
  </mx:VBox>
</mx:Panel>
```

代码 14.14 将 Box 组件的 backgroundColor 属性绑定到 HSlider 组件的 value 属性，运行的效果是在 Box 组件中实时显示 HSlider 组件的拖动值。dataTipFormatFunction 属性指定了 HSlider 组件拖动显示的提示信息格式。

（5）在代码 14.14 最下方 Label 组件的 text 属性中，绑定的值使用字符串连接符"+"将 strColor 变量的内容显示出来。strColor 变量是在 ActionScript 脚本中声明的一个可绑定变量，在文件中添加代码 14.15 实现绑定代码。

代码 14.15　实现绑定 ActionScript 脚本

```
 <mx:Script>
    <![CDATA[
    [Bindable]
    private var strColor:String="#000000";
        private function intToHex(color:int = 0):String {
            var mask:String = "000000";
            var str:String = mask + color.toString(16).toUpperCase();
            strColor="#" + str.substr(str.length - 6);
            return strColor;
        }
    ]]>
</mx:Script>
```

（6）运行程序，图 14-7 所示为初始化时的效果。Box 组件和 Label 组件都绑定到 HSlider 组件，Box 组件显示 HSlider 组件对应颜色，Label 组件显示 HSlider 组件对应的颜色值。

拖动 HSlider 组件中的滑块改变其位置，此时在 Box 组件和 Label 组件中会实时显示相对应的颜色和颜色值，如图 14-8 所示。

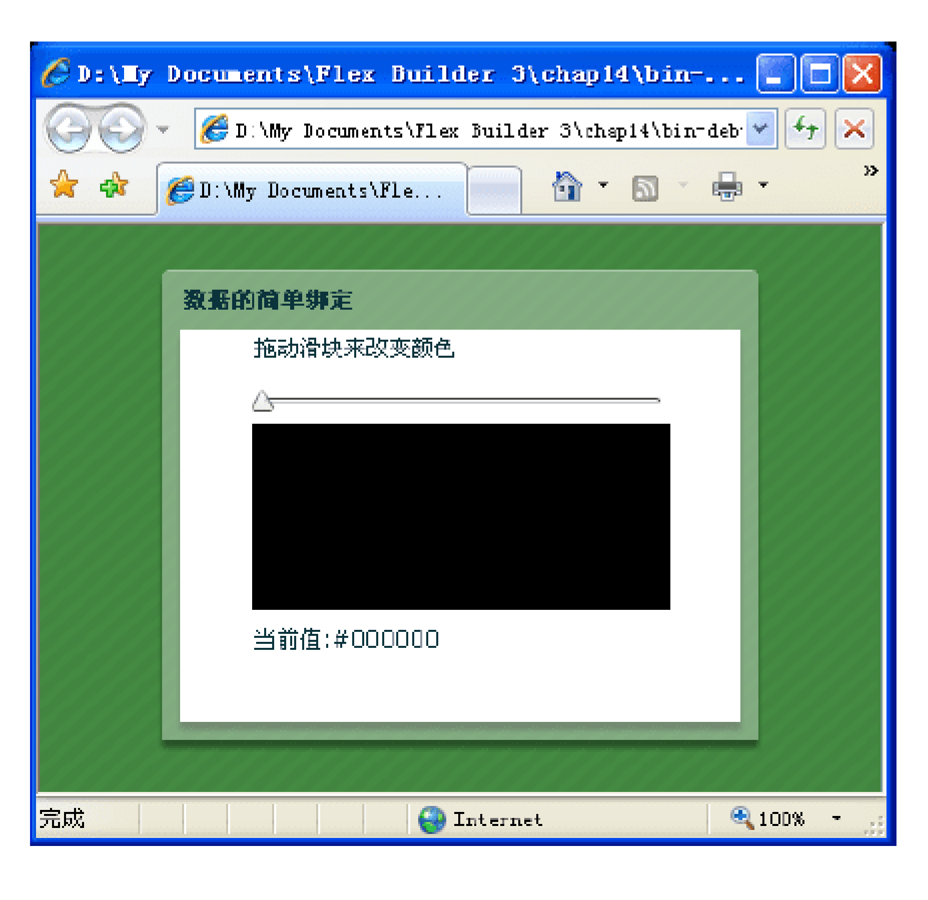




图 14-7　初始化时

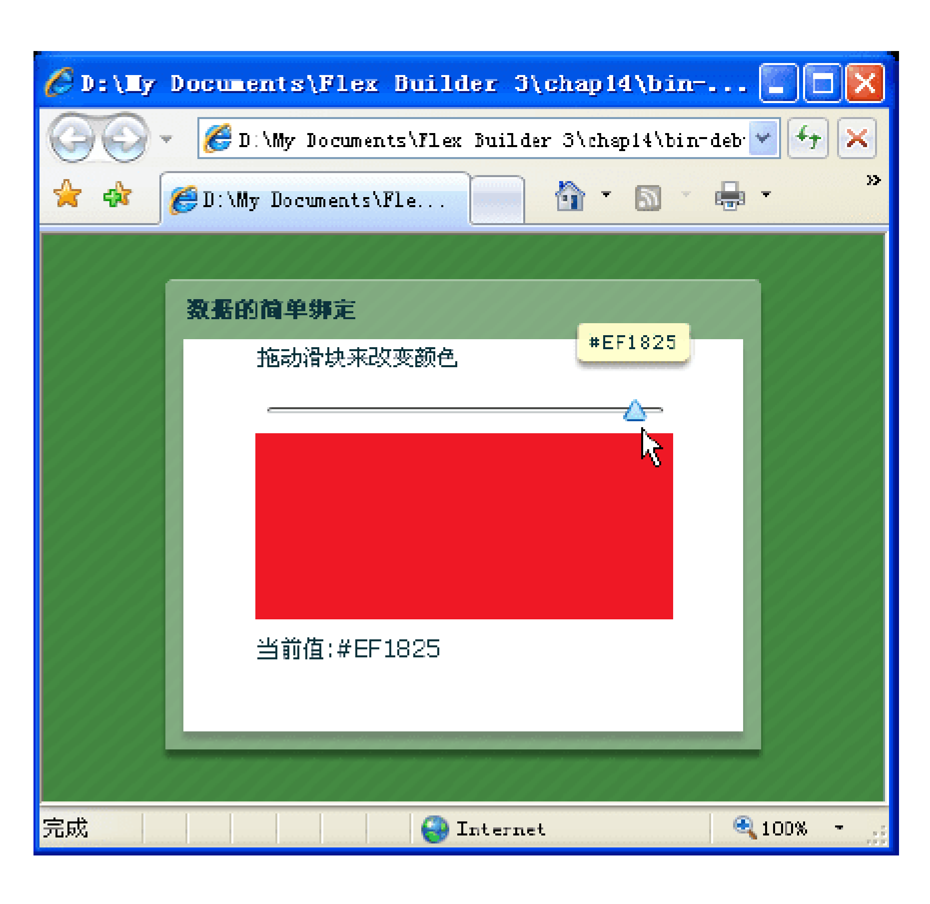


图 14-8　拖动时

本程序中使用“{}”绑定方式，从而有效地减少了代码。若不使用绑定方式，本程序必须使用事件处理：在<mx:HSlider>组件的拖动值发生改变时，修改<mx:Text>组件的 text 值。另外需要注意的是，绑定时类型要一致。text 属性的值为字符串类型，除了字符串类型外，不能绑定别的数据类型。[Bindable]指定变量为绑定，这样当拖动值发生改变时，能够及时响应到字符串中。

使用大括号运算符“{}”是绑定中最简单，也是最直观、使用最多的方式。下面，通过一个实例来了解一下该运算符的其他绑定方式。创建一个新的 MXML Application 文件，命名为 baseBind.mxml，并添加如代码 14.16 所示的代码。

代码 14.16　使用“{}”绑定方式

```
<mx:Model id="myModel">
    <myModel>
        <!--直接绑定方式，单个值-->
        <Pname>{nameInput.text}</Pname>
        <!-- 绑定时使用字符串 -->
        <Pdesc>New~ {nameInput.text}</Pdesc>
        <!--绑定时执行计算 -->
        <Pprice>{(Number(numberInput.text) as Number) * 0.8}</Pprice>
        <!-- 绑定时使用表达式 -->
        <Pmore>{(isShow.selected) ?"发布到首页" : "发布到仓库" } {nameInput.
        text}</Pmore>
    </myModel>
</mx:Model>
<mx:Panel paddingBottom="10" paddingLeft="10" paddingRight="10" paddingTop=
```

```
"10"
    width="100%" height="100%"  title="简单绑定方式">
    <mx:Form>
        <mx:FormItem label="产品名称: ">
            <mx:TextInput id="nameInput" width="275"/>
        </mx:FormItem>
        <mx:FormItem label="高级选项: ">
            <mx:RadioButton  id="isShow1" label="不显示(默认)" groupName="home"
            selected="true" />
            <mx:RadioButton  id="isShow" label="首页显示" groupName="home" />
        </mx:FormItem>
        <mx:FormItem label="产品价格: ">
        <mx:TextInput id="numberInput" text="0" width="275"/>
        </mx:FormItem>
    </mx:Form>
    <mx:Label text="{'产品名称: '+myModel.Pname}"/>
    <mx:Label text="{'产品描述: '+myModel.Pdesc}"/>
    <mx:Label text="{'折扣价格: '+numberInput.text+'*0.8='+myModel.Pprice}"/>
    <mx:Label text="{'详细信息: '+myModel.Pmore}"/>
</mx:Panel>
```

代码 14.16 比较简单，使用了前面介绍过的<mx:Model>组件作为数据模型，在数据模型中进行各种绑定。运行程序，输入一些内容，查看绑定效果如图 14-9 所示。

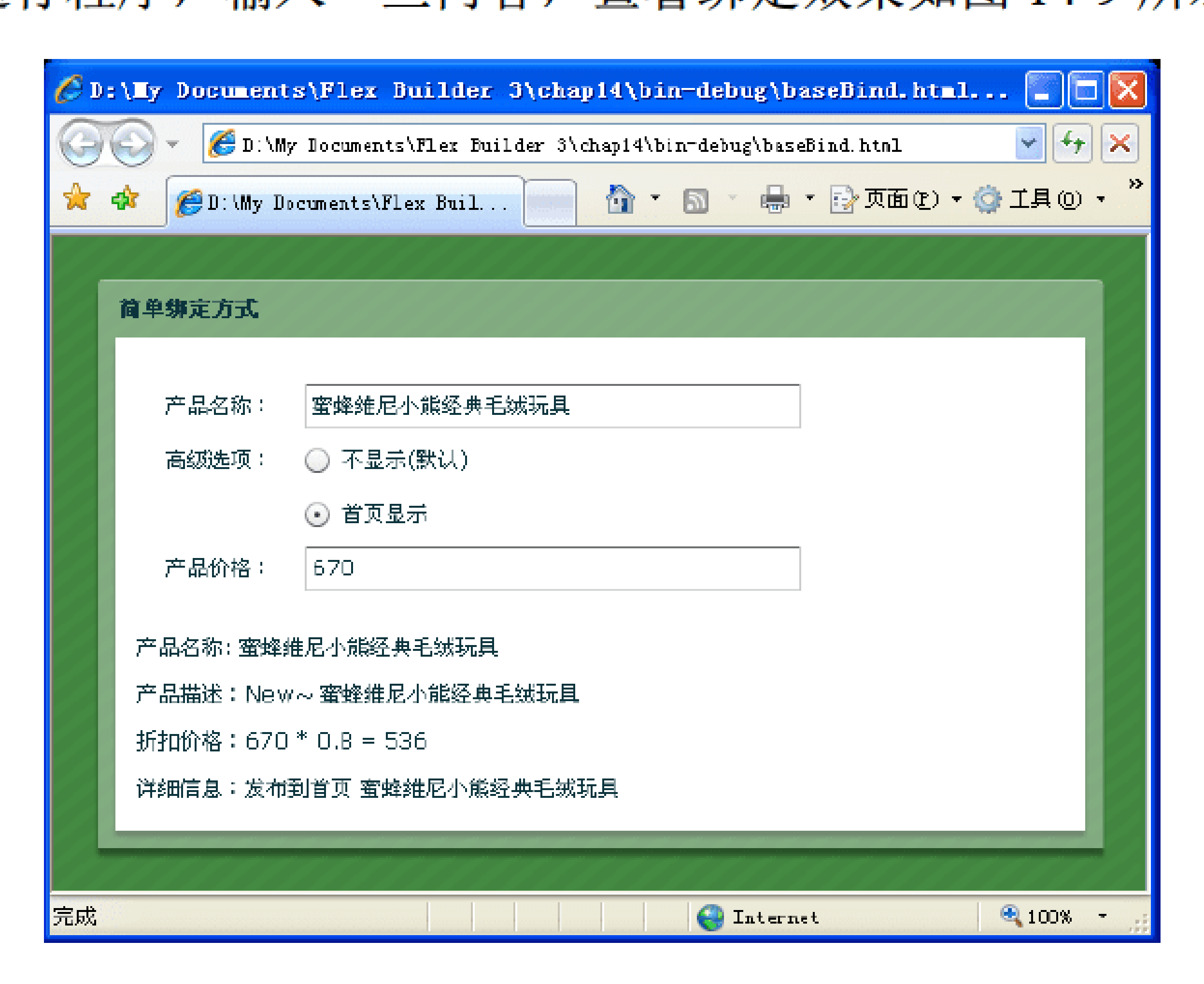


图 14-9　绑定效果

14.2.2　使用<mx:Binding>组件

<mx:Binding>组件也可实现数据绑定，其语法如下所示。

```
<mx:Binding source="绑定源"  destination="绑定目标"/>
```

其中，source 属性指明被绑定的源，destination 属性指明绑定的目标。需要注意的是两者指向的类型必须相同。例如，source 属性为<mx:Text>组件的 text 属性，destination 属性为<mx:HSlider>组件的 value 属性，两者都为字符串属性。

使用“{}”方式的绑定同样可以使用<mx:Binding>组件来完成。下面介绍使用<mx:Binding>组件实现上节图 14-9 所示的效果，步骤如下。

首先，新建一个名为 mx:Binding.mxml 的 MXML Application 文件。在文件中添加<mx:Model>数据模型，它与代码 14.16 中相同。

要使用<mx:Binding>组件进行绑定必须指定 source 和 destioination 属性。这就要求对 Panel 组件中显示结果的 Label 组件进行修改，代码 14.17 所示为修改后代码。

代码 14.17　修改 Panel 组件

```
<mx:Panel paddingBottom="10" paddingLeft="10" paddingRight="10" paddingTop=
"10"
    width="100%" height="100%"  title="&lt;mx:Binding&gt;组件绑定">
    <mx:Form>
        <mx:FormItem label="产品名称: ">
            <mx:TextInput id="nameInput" width="275"/>
        </mx:FormItem>
        <mx:FormItem label="高级选项: ">
            <mx:RadioButton  id="isShow1" label="不显示(默认)" groupName="home"
            selected="true" />
            <mx:RadioButton  id="isShow" label="首页显示" groupName="home" />
        </mx:FormItem>
        <mx:FormItem label="产品价格: ">
        <mx:TextInput id="numberInput" text="0" width="275"/>
        </mx:FormItem>
    </mx:Form>
    <mx:Label id="lblOne" />
    <mx:Label id="lblTwo" />
    <mx:Label id="lblThree"/>
    <mx:Label id="lblFour"/>
</mx:Panel>
```

接下来，添加<mx:Binding>组件实现从<mx:Model>数据模型中将值绑定到相应 Label 组件，如代码 14.18 所示。

代码 14.18　使用<mx:Binding>组件进行绑定

```
<mx:Binding source="{'产品名称: '+myModel.Pname}" destination="lblOne.text" />
<mx:Binding source="{'产品描述: '+myModel.Pdesc}" destination="lblTwo.text" />
<mx:Binding source="{'折扣价格: '+numberInput.text+' * 0.8 = '+myModel.Pprice}"
destination="lblThree.text" />
<mx:Binding source="{'详细信息: '+myModel.Pmore}" destination="lblFour.text" />
```

现在，保存对 mx:Binding.mxml 文件的修改并运行，输入一些内容查看效果与图 14-9 相同。

14.2.3　使用 ActionScript 脚本

经过前两节的学习，了解了数据绑定的一般用法。在实际开发中，经常会遇到比这些应用更复杂的情况。本节继续讲解数据绑定的用法。

除了在 MXML 中使用大括号和<mx:Binding>组件来定义数据绑定外，Flex 还允许用户在 ActionScript 脚本中定义数据绑定。像前面介绍的[Bindable]关键字，就是 ActionScript 脚本绑定数据的最简单方式。下面介绍另一种方式，通过使用 BindingUtils 类来实现。

该类位于 mx.binding.utils 包中，包含两个用于处理动态数据绑定的静态方法。

- **bindProperty()**

用来进行属性级别的绑定，方法有 5 个参数，依次为：site 参数，表示目标对象；prop 参数，表示目标对象的公有属性名，当值发生变化时，执行绑定事件；host 参数，指定数据源对象；chain 参数，表示数据源对象被绑定的属性名，或者通过 getter 定义的函数名；commitOnly 参数，默认值为 false，表示只要发生值的改变就会触发绑定，如果为 true，表示对对象有写操作时仍然会执行绑定事件，但不会触发绑定行为，只有确认了改变的数据，并执行事件 valueCommit 后，绑定行为才开始执行。

- **bindSetter()方法**

这个方法仅用于 setter 和 getter 函数的绑定，有 4 个参数，setter 参数表示一个函数名，用来改变数据源对象的值；host 参数是一个数据源对象；chain 参数和 commitOnly 参数的含义与 bindProperty()方法中相同。

这两个方法中的 commitOnly 参数主要针对那些被频繁修改的变量，例如输入文本，只有当文本确认后，才触发事件，其他情况下很少使用。

下面仍以 14.2.1 节代码 14.16 所实现的功能为例，将其使用 BindingUtils 类实现。先新建一个新的 MXML Application 文件，再添加代码 14.17 中的 Panel 组件及<mx:Model>数据模型。

接下来对 MXML 的 Application 标签进行修改，添加“creationComplete="initApp()"”，使应用程序在载入完成后执行 initApp()函数。initApp()是一个 ActionScript 函数，它使用 BindingUtils 类的 bindProperty()方法实现绑定，但在使用之前需要先引用“mx.binding.utils.BindingUtils”类。这段代码如 14.19 所示。

代码 14.19　使用 BindingUtils 类绑定

```
<mx:Script>
    <![CDATA[
        import mx.binding.utils.BindingUtils;

        private function initApp():void
```

```
            {
            BindingUtils.bindProperty(lblOne,"text",myModel,"Pname");
            BindingUtils.bindProperty(lblTwo,"text",myModel,"Pdesc");
            BindingUtils.bindProperty(lblThree,"text",myModel,"Pprice");
            BindingUtils.bindProperty(lblFour,"text",myModel,"Pmore");
            }
        ]]>
    </mx:Script>
```

保存文件并执行程序，在浏览器输入一些内容查看效果，观察与前面两种实现方法的不同之处。

14.3　DataGird 组件

DataGrid 组件的作用与 HTML 页面中的表格类似，可以将数据以行、列的格式显示出来。DataGrid 是 Flex 中较为复杂的一个组件，它具有很多强大的功能，主要包括：每一列的宽度不固定，用户可以在运行时调整宽度；用户可以在运行时调整列的顺序；单击列标题对列中的数据进行排序；可以自定义每列的标题；还允许用户自定义每个单元格中的显示内容和模板等。

14.3.1　显示数据

显示数据列表是 DataGrid 组件最基本的应用，只有把数据显示出来，才可以实现其他的功能。下面通过实例来演示 DataGrid 在显示数据上的便利。

在使用 DataGrid 组件之前，首先新建一个 MXML Application 文件并在其中定义要显示的数据。14.1 节介绍了各种数据模型的定义，这里使用<mx:XML>组件显示数据的来源。添加代码 14.20 的内容到页面上。

代码 14.20　定义数据

```
<mx:XML id="xmlModel" format="e4x">
<UserList>
<User>
    <ID>E5091</ID>
    <Name>王梦屹</Name>
    <Email>ayhncn@163.com</Email>
    <Career>销售主管</Career>
    <Site>www.itzcn.net</Site>
</User>
<User>
    <ID>I9395</ID>
    <Name>祝红涛</Name>
```

```
        <Email>somboy@126.com</Email>
        <Career>软件开发工程师</Career>
        <Site>www.itzcn.com</Site>
    </User>
    <User>
        <ID>C0892</ID>
        <Name>贺强</Name>
        <Email>Qiange@188.com</Email>
        <Career>市场策划员</Career>
        <Site>www.webzcn.cn</Site>
    </User>
    </UserList>
    </mx:XML>
```

代码 14.20 定义了一个 xmlModel 数据模型，其中以 XML 格式保存了 3 条数据。接下来，从 Components 窗格中添加一个 DataGrid 组件到页面，默认会包括 3 列，这里要根据 xmlModel 中的数据进行修改。在 DataGrid 组件中的 columns 标签内包括了所有列，每一列对应一个 DataGridColumn 标签。

DataGridColumn 标签有两个与数据显示相关的重要属性。HeaderText 属性定义列的标题名称，dataField 属性定义数据源中每条数据包含的属性名。要从数据模型中显示数据，dataField 属性是必须的，否则数据无法显示。

现在根据 xmlModel 数据模型的定义，对默认的 DataGird 组件进行修改，代码 14.21 所示为修改后的布局。

代码 14.21　添加 DataGrid 组件

```
     <mx:Panel width="95%"  title="查看联系人列表" >
      <mx:DataGrid width="100%" dataProvider="{xmlModel.User}">
        <mx:columns>
          <mx:DataGridColumn headerText="编号" dataField="ID" />
          <mx:DataGridColumn headerText="姓名" dataField="Name"/>
          <mx:DataGridColumn headerText="邮箱" dataField="Email"/>
          <mx:DataGridColumn headerText="职业" dataField="Career"/>
          <mx:DataGridColumn headerText="网站" dataField="Site"/>
       </mx:columns>
      </mx:DataGrid>
    </mx:Panel>
```

在代码 14.21 中，dataProvider 属性用于指定 DataGrid 组件中使用的数据来源，这里使用“{xmlModel.User}”表示绑定到 xmlModel 数据类型中的 User 节点。往下的 dataField 属性值和 xmlModel 中 User 节点的子节点相同。也就是说，这些列显示的数据来自 User 子节点的定义。

保存文件并执行程序，运行效果如图 14-10 所示。在 columns 标签中定义的每一列都以指定的 headerText 和 dataField 显示出来了。

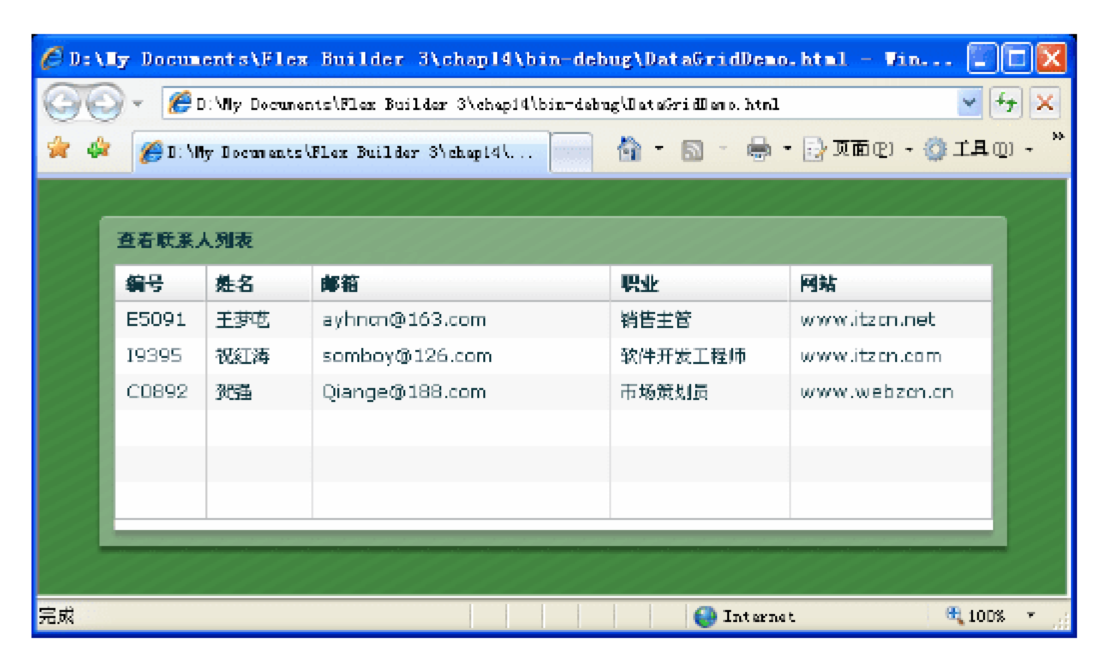
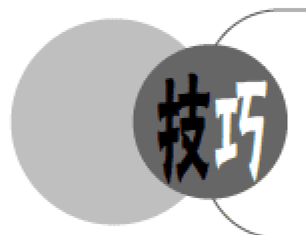


图 14-10　DataGrid 组件显示数据

14.3.2　获取行数据

通过上节的简单操作，不用编写任何代码即可实现在 DataGrid 组件上显示数据，可见 DataGrid 组件显示数据的方法非常简单。另外，此程序运行后，还可以调整列的顺序、列的宽度，如果单击列的标题，还可以对该列进行排序，这些操作都由 DataGrid 组件自动完成。



通过为 DataGridColumn 标签指定 width 属性可以自定义列在运行后的默认宽度。visible 属性可指定列在运行时是否可见。

DataGrid 组件有一个 selectedItem 属性，该属性返回代表当前选中行的数据，经常在数据绑定中使用。当它的值发生变化时，也就是用户单击选中某行时，使用该属性的任何组件也会自动更新。

例如，在上面代码 14.21 的 DataGrid 组件下添加代码 14.22 来实时跟踪显示 DataGrid 组件中选中行的信息。每当选中行发生变化时，标签的文字都会相应改变，如图 14-11 所示。

代码 14.22　显示选中行数据

```
<mx:Label text="姓名: {dgUser.selectedItem.Name}" />
<mx:Label text="职业: {dgUser.selectedItem.Career}" />
<mx:Label text="网站: {dgUser.selectedItem.Site}" />
```

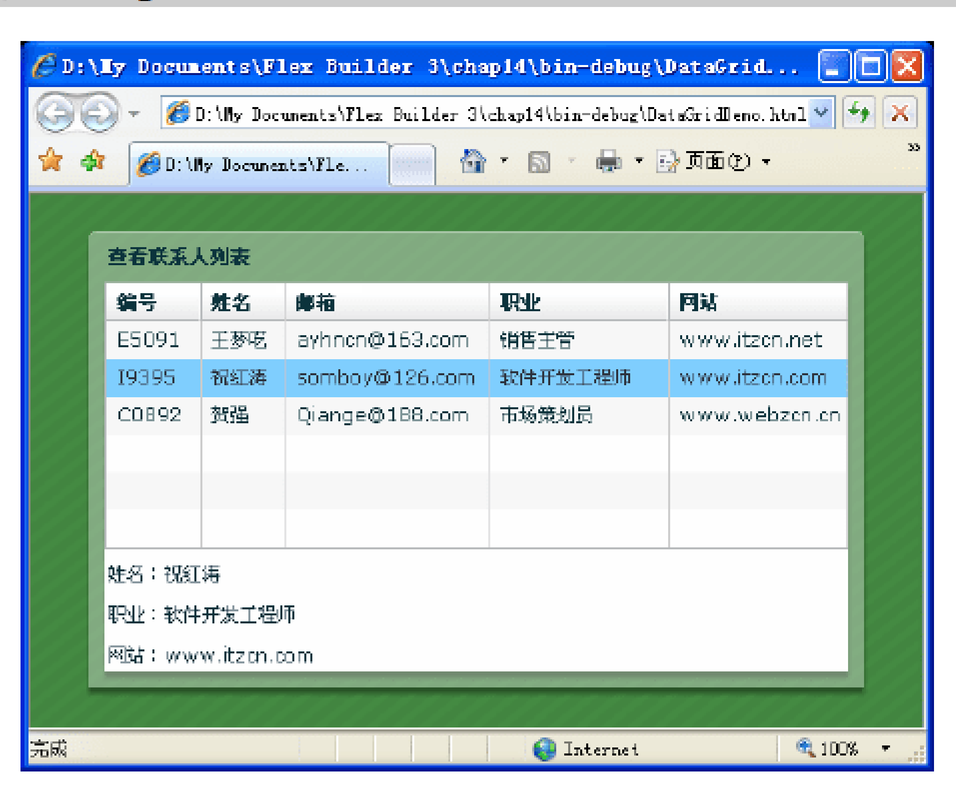


图 14-11　查看选中行数据

在添加如代码 14.22 所示的代码后，执行程序之前还需要先为 DataGrid 组件添加一个 id 属性。根据所示得知，id 的值为 dgUser。

14.3.3 自定义列

在使用 DataGrid 组件时大量的数据以行和列的形式显示出来。在这些数据中，可能有很多不同的类型，这就需要在显示时加以区分。例如，对于日期类型数据可以在显示时添加分隔符"-"等。

下面以上节的例子为基础，通过修改少量的代码在 DataGrid 组件中为“网站”列的数据自定义一种显示格式，即添加前缀字符串“http://”，步骤如下。

首先打开上节保存的实例文件。找到 DataGrid 组件的代码为“网站”列添加一个 labelFunction 属性。该属性可以指定处理列数据格式的函数名称。代码 14.23 所示为修改后的列。

代码 14.23　为“网站”列添加 labelFunction 属性

```
<mx:DataGridColumn headerText="网站" dataField="Site" labelFunction="Site-
Format"/>
```

如代码 14.23 所示，SiteFormat 为 ActionScript 函数。在文件中添加<mx:Script>再编写实现代码，如 14.24 所示。

代码 14.24　SiteFormat()函数

```
<mx:Script>
    <![CDATA[
        import mx.controls.dataGridClasses.DataGridColumn;
        private function SiteFormat(sItem:Object,Site:DataGridColumn):String
        {
            return "http://"+sItem.Site;
        }
    ]]>
</mx:Script>
```

SiteFormat()函数接受两个参数，调用的第 1 个参数是 sItem，它的类型必须是 Object，表示一行数据；第 2 个参数是想要进行格式化的列的名称，在这个例子中为 Site，它的类型必须是 DataGridCoumn。最后，函数的返回值必须为 String。

运行应用程序，在 DataGrid 组件中的“网站”列都多出了“http://”，如图 14-12 所示。由 DataGrid 组件下方的结果可以看出，实际存储的数据并未包含“http://”。也就说明，这是由 DataGrid 组件为指定列添加的，并未对原始数据进行修改。

接下来学习如何对显示日期的列进行格式化。如图 14-12 所示，DataGrid 组件中未包含任何日期列，也没有适合作为日期显示的列。因此，首先需要在 DataGrid 组件中增加一个日期列，如代码 14.25 所示。

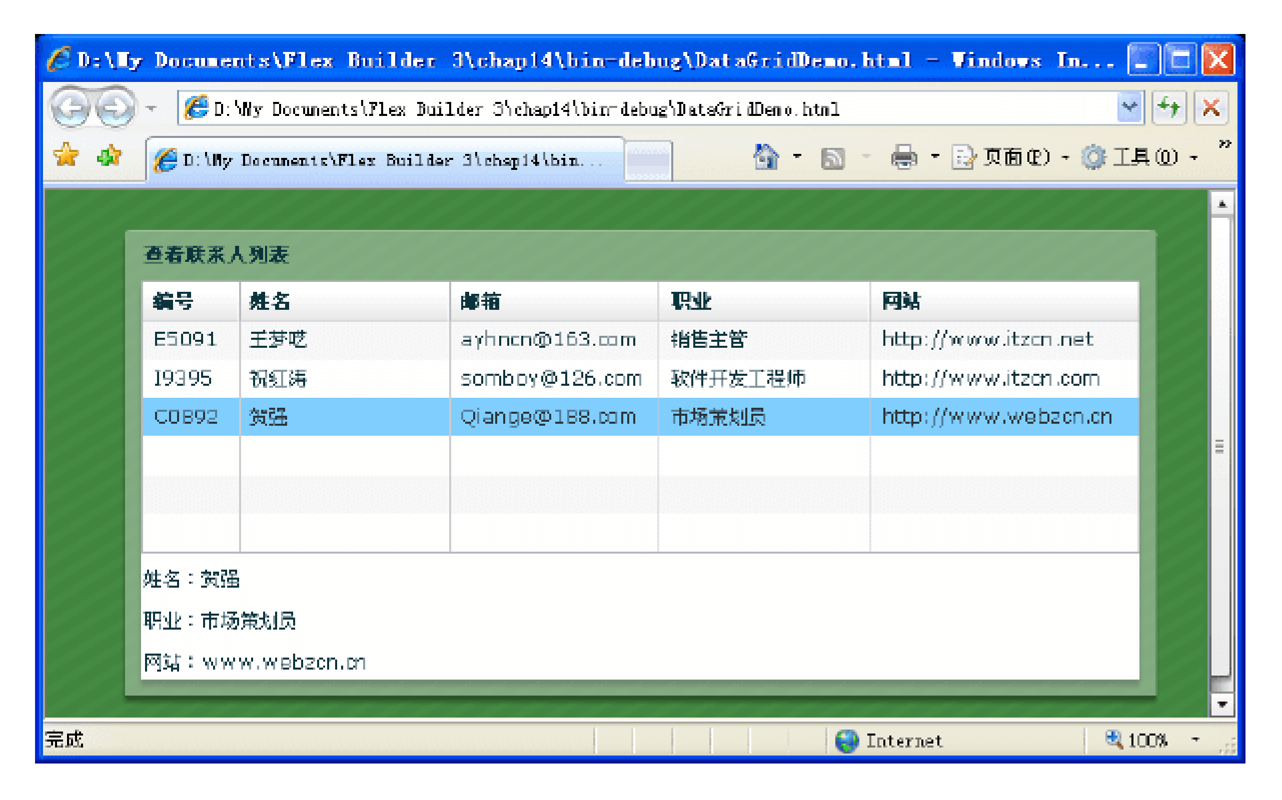


图 14-12　格式化“网站”列的效果

代码 14.25　添加日期列

```
<mx:DataGridColumn headerText="更新日期" labelFunction="eFormat"/>
```

从代码 14.25 中可以看出，在定义列时同时指定了 labelFunction 属性。该属性并不陌生，在格式化“网站”列时使用过。

现在，在 ActionScript 中添加 eFormat()函数，实现对“更新日期”列的格式化。这里要注意一点：虽然在代码 14.25 中没有指定 dataField 属性，但是 labelFunction 属性指定的 eFormat()函数仍然必须具有两个参数。代码 14.26 所示为最终 eFormat()函数的代码。

代码 14.26　eFormat()函数

```
private function eFormat(obj:Object,eTime:DataGridColumn):String
{
    var myDateFormatter:DateFormatter=new DateFormatter();
    myDateFormatter.formatString="YYYY 年 M 月 D 日";
    return myDateFormatter.format('12/25/2008');
}
```

这里使用 DateFormatter 类来指定日期格式，并完成格式化操作。在学习完本章之后，读者也可以使用日期格式化组件来实现同样的功能。图 14-13 所示为运行效果。

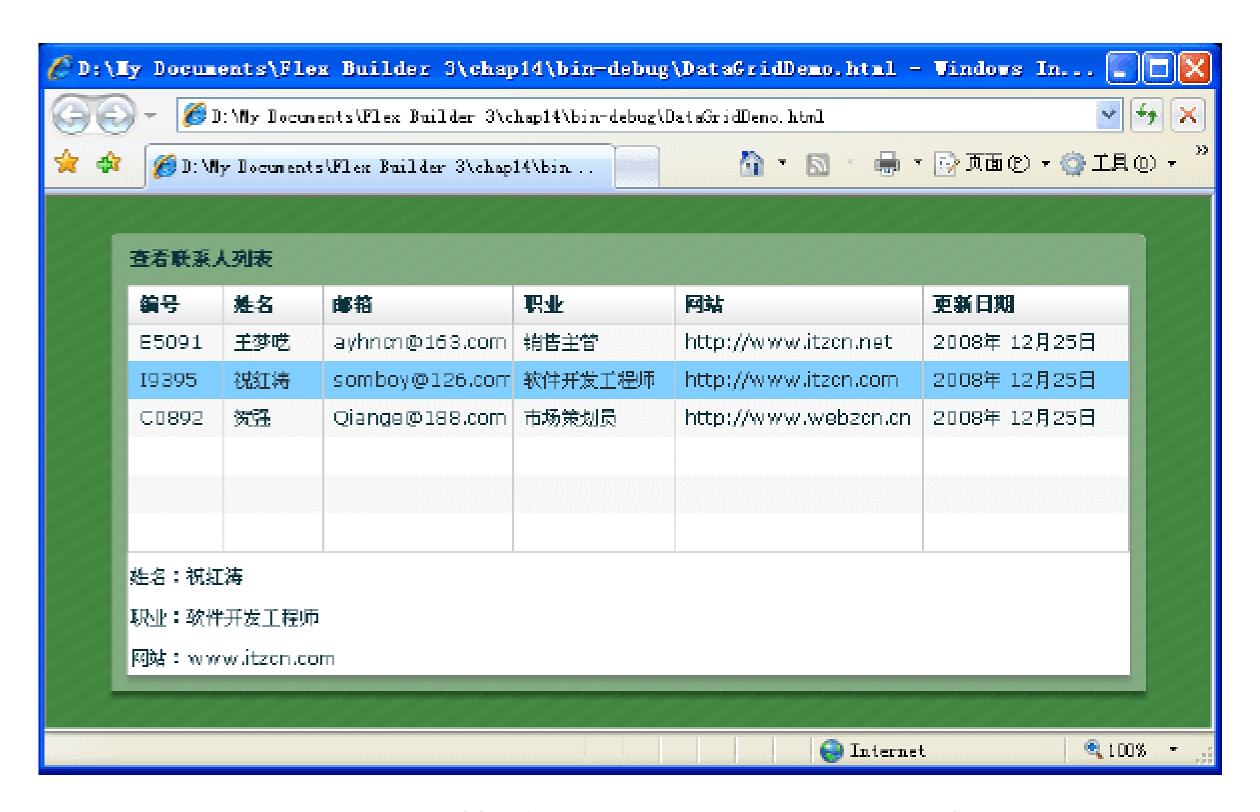


图 14-13　格式化“更新日期”列效果

14.3.4 编辑数据

DataGrid 组件除了在数据显示上为用户提供了简单的操作外，还可以对显示的数据进行编辑。也就是说，允许用户在行中编辑已有的数据或者输入新的数据。下面将演示如何让“邮箱”列变得可以编辑。

（1）首先，在 MXML Application 文件中为 DataGrid 组件添加 editable 属性，该属性是 1 个布尔值用于控制是否启用数据的可编辑性，设置为 true。

（2）现在运行程序，单击列中的单元格，可以看到，在 DataGrid 组件中可以更改任意列的数据，如图 14-14 所示。这里只需要对“邮箱”列可编辑，因此必须对 DataGrid 组件的代码进行修改，如代码 14.27 所示。

代码 14.27　设置“邮箱”列为可编辑列

```
<mx:DataGrid width="100%" dataProvider="{xmlModel.User}" id="dgUser"
editable="true">
   <mx:columns>
     <mx:DataGridColumn headerText="编号" dataField="ID" width="60" editable=
     "false"/>
     <mx:DataGridColumn headerText="姓名" dataField="Name" editable="false"/>
     <mx:DataGridColumn headerText="邮箱" dataField="Email" editable="true"/>
     <mx:DataGridColumn headerText="职业" dataField="Career" editable=
     "false"/>
     <mx:DataGridColumn headerText="网站" dataField="Site" labelFunction=
     "SiteFormat" editable="false"/>
     <mx:DataGridColumn headerText="更新日期" labelFunction="eFormat" editable=
     "false"/>
  </mx:columns>
 </mx:DataGrid>
```

（3）再次运行程序，在 DataGrid 组件中其他列的单元格内单击，发现不能编辑，而仅能对“邮箱”列的数据进行编辑，效果如图 14-15 所示。

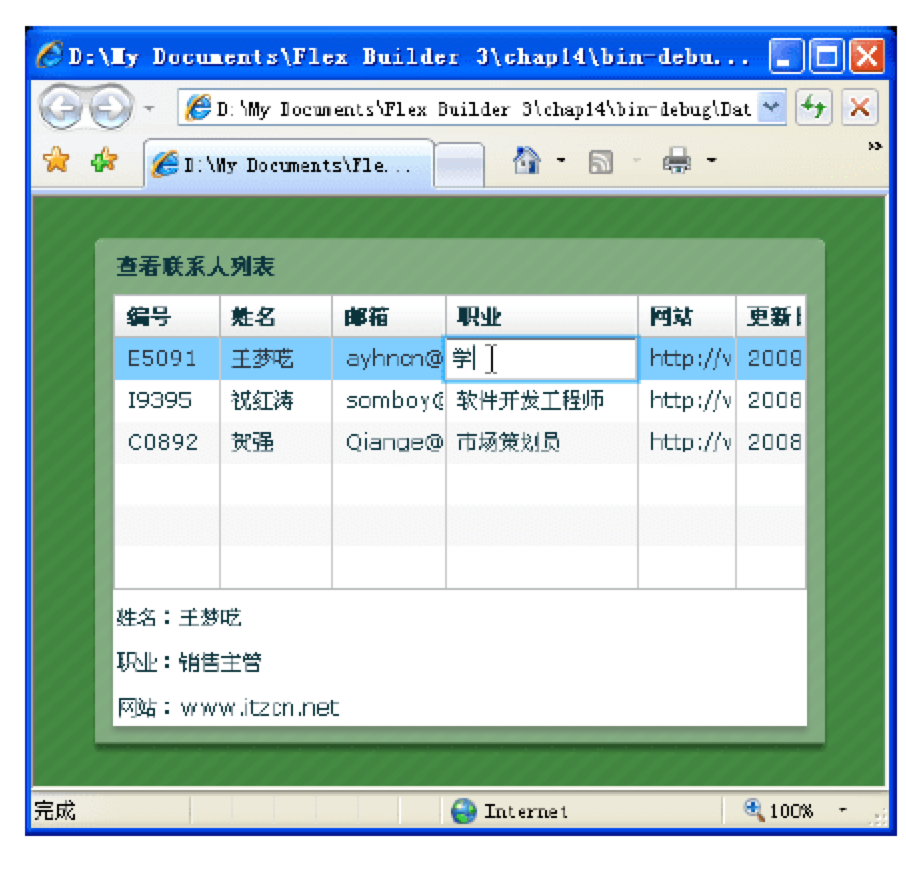


图 14-14　editable 属性为 true 效果

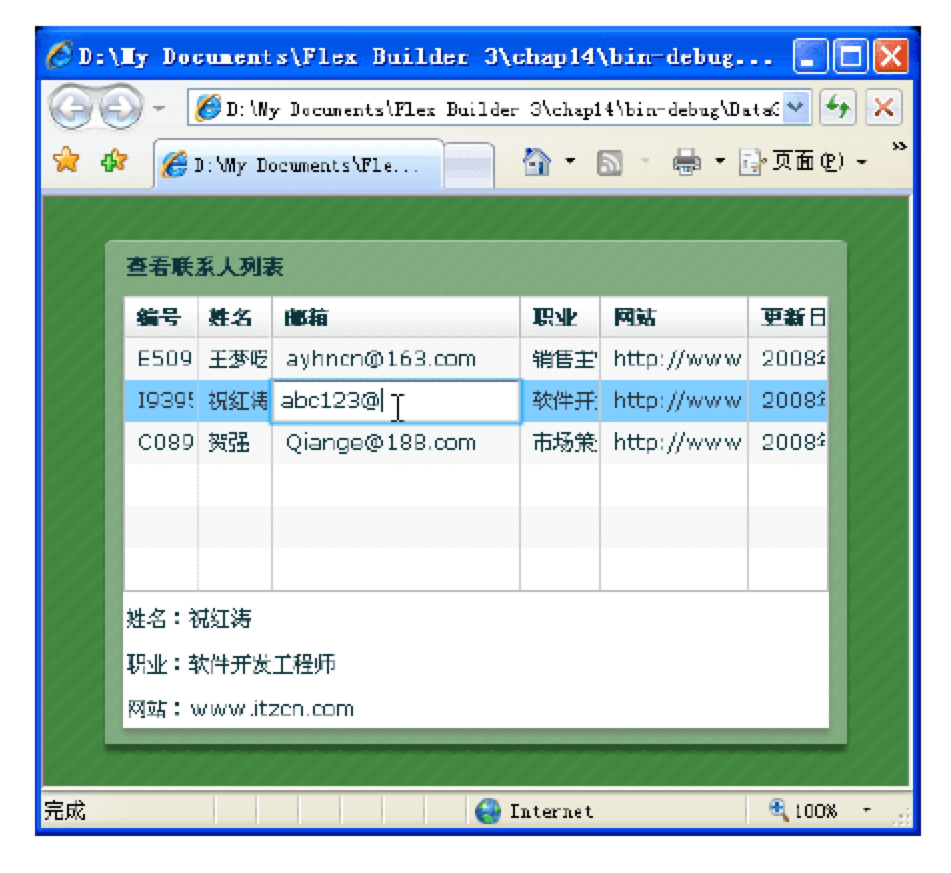


图 14-15　编辑“邮箱”列效果

（4）接下来换个思路，使用 DataGrid 组件可编辑数据特性，实现更强的功能。更改一下 DataGrid 组件的结构，添加一个“备注”列允许用户输入对该行数据的描述信息，如代码 14.28 所示。

代码 14.28　添加“备注”列

```
<mx:DataGridColumn headerText="备注" dataField="MoreInfo" editable="true" />
```

（5）运行程序测试一下代码。运行后，产生了一个新列，单击该列的单元格用户可以输入任何的信息。

现在，假设为了允许用户输入备注信息，需要把 TextArea 组件放在单元格中。要实现这个功能有两种方法：使用 itemEditor 属性或者 itemRenderer 属性。这两个属性都允许将控件嵌入到单元格中，不过两者的功能有些区别，但无论使用哪个属性，前提是单元格必须可编辑。下面分别介绍这两种方法，具体步骤如下。

首先为新增的“备注”列添加 itemEditor 属性值为 TextArea 组件，如代码 14.29 所示。

代码 14.29　使用 itemEditor 属性

```
<mx:DataGridColumn headerText="备注" dataField="MoreInfo" editable="true"
itemEditor="mx.controls.TextArea"/>
```

在代码 14.29 中不仅指明了要把 TextArea 组件嵌入单元格，而且定义了包含它的包。这样，将导入操作和组件合并到一个简单的 MXML 表达式中。

运行程序，itemEditor 属性在页面加载后的效果与普通的相同。但在单击“备注”列的单元格后，会看到 TextArea 组件，并允许输入内容，如图 14-16 所示。

接下使用 itemRenderer 属性，将“备注”列修改为代码 14.30 所示的内容。

代码 14.30　使用 itemRenderer 属性

```
<mx:DataGridColumn headerText="备注" dataField="MoreInfo" editable="true"
itemRenderer="mx.controls.TextArea"/>
```

此时运行程序，应该可以看到结果与 itemEditor 属性明显不同。每一行的“备注”列都被打开了并以 TextArea 组件形式呈现，以方便用户的输入，如图 14-17 所示。

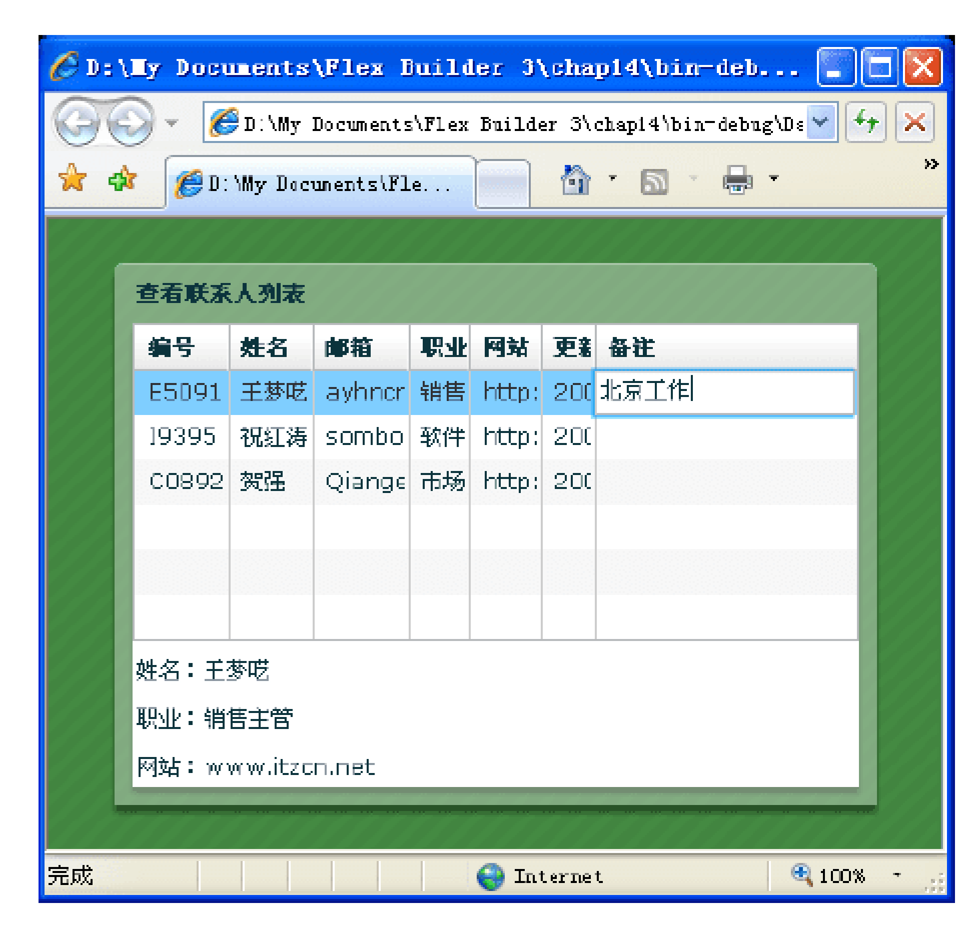


图 14-16　itemEditor 属性效果

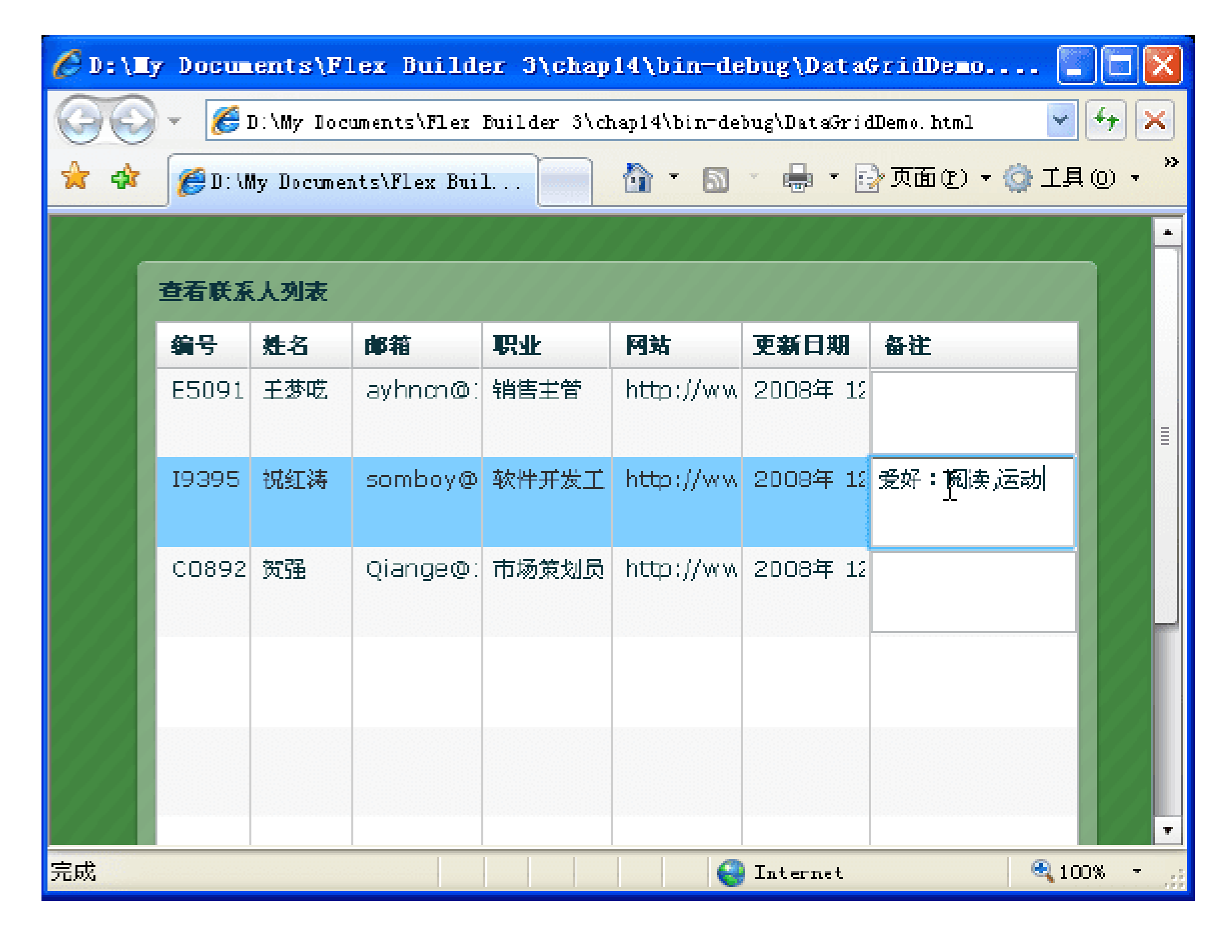


图 14-17　itemRenderer 属性效果

上面介绍的方法，一次只能在单元格中嵌入一个组件。通过下面介绍的方法，可以在单元格中嵌入一个自定义的复合组件。步骤如下所示。

（1）选择 File | New | MXML Component 命令，打开 New MXML Component 对话框。

（2）在 Filename 文本框中输入 MoreInfoForm，从 Based on 下拉列表框中选择 Form 项，如图 14-18 所示。

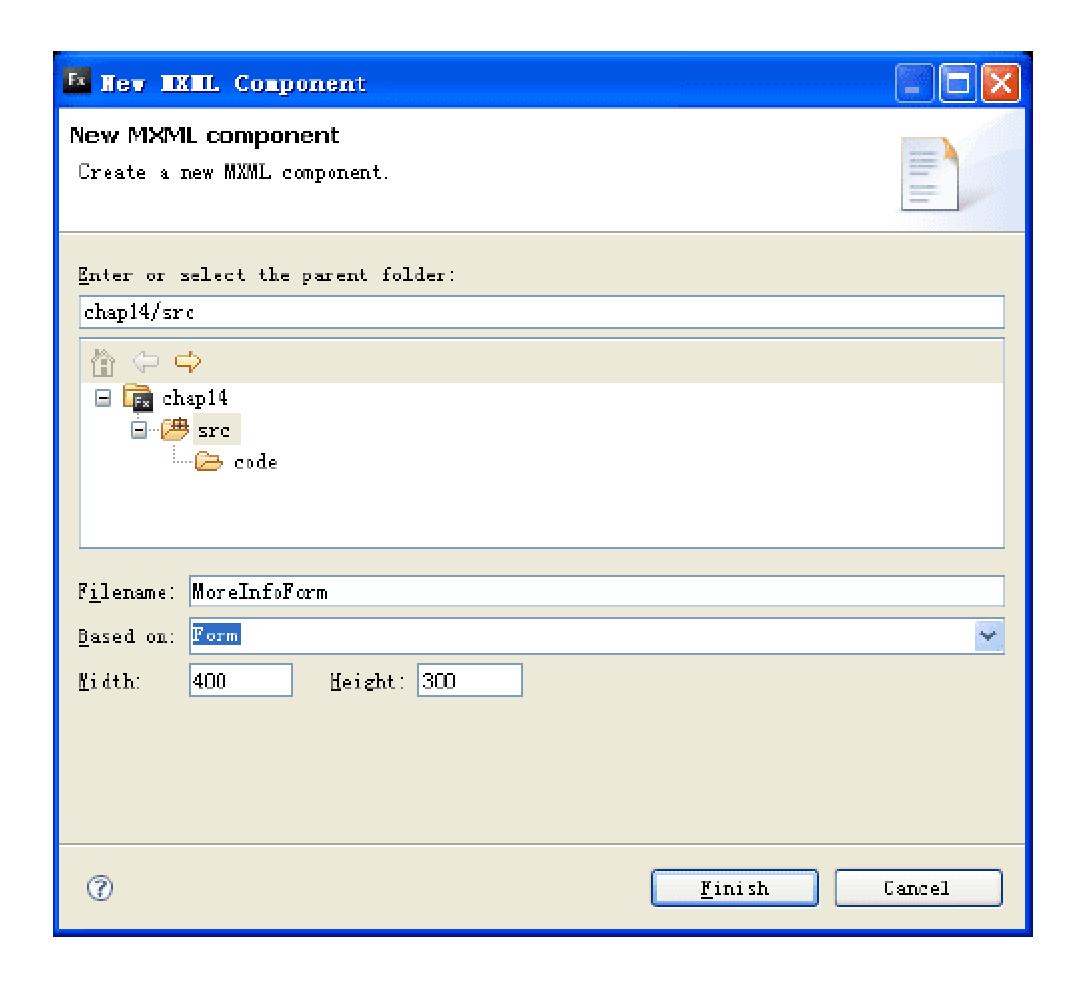
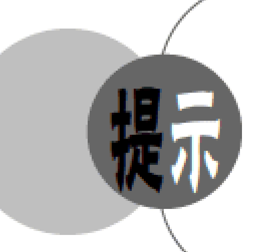


图 14-18　New MXML Component 对话框



在第 13 章中，已经学习过如何自定义组件，了解到每个自定义组件必须以一个容器为基础进行扩展，但是该容器不可以是 Application，在本例中选择了 Form 组件。

（3）单击 Finish 按钮，再切换到 Source 视图下，依次添加两个 TextInput 组件分别用于输入“地点”和“爱好”项。代码 14.31 所示为最终编辑后的组件代码。

代码 14.31　自定义 MoreInfoForm 组件代码

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Form xmlns:mx="http://www.adobe.com/2006/mxml"  >
    <mx:FormItem label="地点: ">
        <mx:TextInput width="80"/>
    </mx:FormItem>
    <mx:FormItem label="爱好: " >
        <mx:TextInput width="80"/>
    </mx:FormItem>
</mx:Form>
```

（4）转换到 Design 视图，使用尺寸调整句柄，对 MoreInfoForm 组件中的空白空间进行调整，调整至合适大小后保存组件。

（5）返回到应用程序的文件。为了让 DataGrid 组件中的每一行内容自动调整其高度，需要添加一个名为 variableRowHeight 的属性，设置值为 true。

（6）接下来的工作就是，让 MoreInfoForm 组件在 DataGrid 组件中进行显示。方法是，更改之前创建的包含有 TextArea 组件的“备注”列，设置其 itemRenderer 属性为 MoreInfoForm。

（7）保存并运行程序，效果如图 14-19 所示。

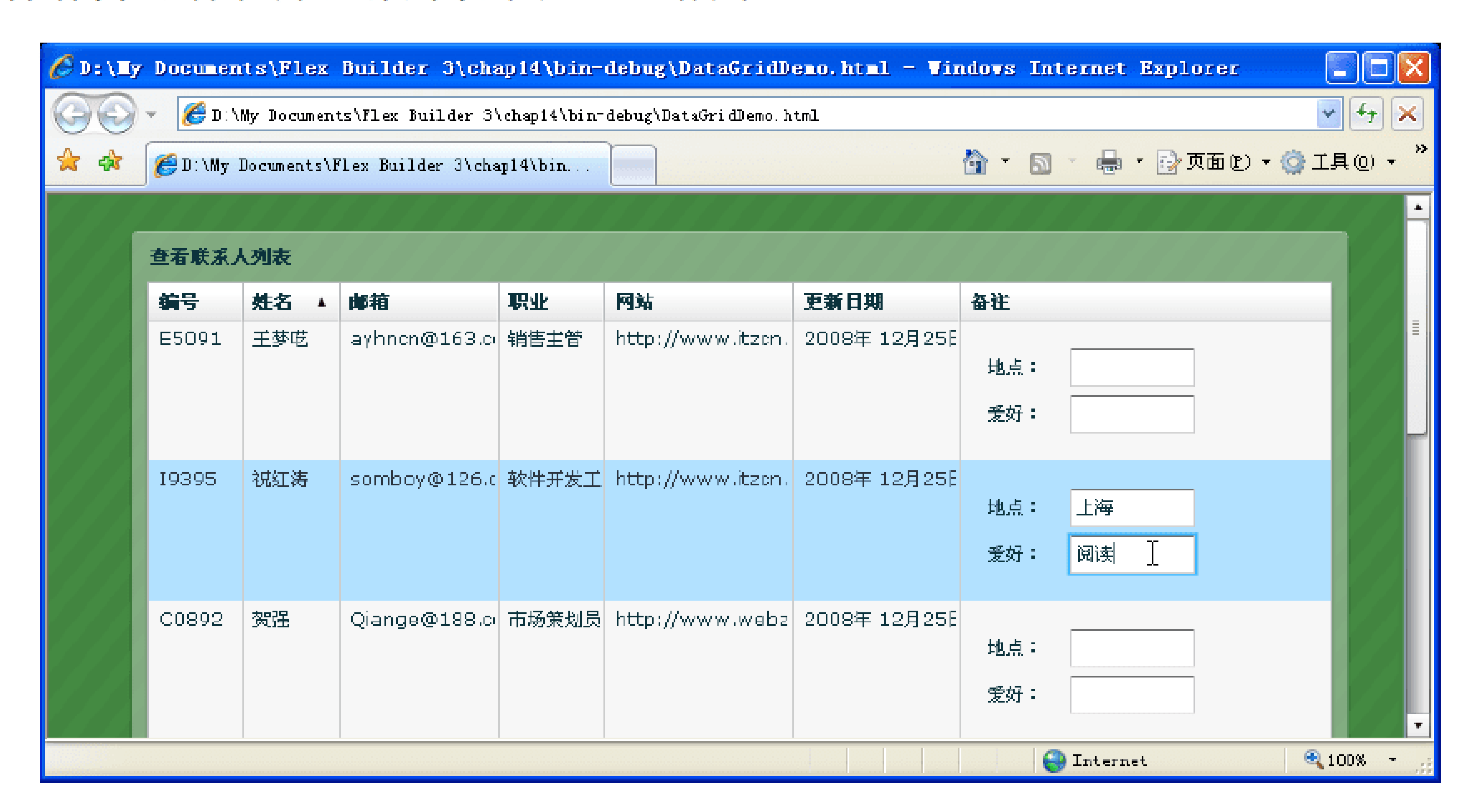


图 14-19　使用自定义组件效果

可以看到，带有多个组件的自定义组件在 DataGrid 组件的单元格中同样可以使用。通过使用这个强大的功能，可以建立各种复杂的情形和编程状况。一种最常用的做法是，创建一个带 Image 组件的自定义组件，其中 Image 组件绑定到 XML 文件中图像的 URL 里，然后再把自定义组件嵌入到 DataGrid 组件的单元格内来显示。

14.4　数据验证

在实际的 Web 应用程序中，可能需要用户提交各式各样的信息，这样就需要进行大量的数据验证。数据验证是指应用程序中对输入的数据进行某种方式的校验。例如，电话号码必须为数字，E-mail 地址有特定的规则等。本节将详细介绍 Flex 3.0 数据验证的基本方法和自定义方法。

14.4.1　数据验证组件概述

在将用户提交的大量信息保存到数据库之前，要对这些用户所输入的信息进行数据的合法性校验，以便后面的程序可以安全顺利地执行。但是如果都需要通过编写程序代码来进行验证的话，将会大大增加开发人员的工作量，并且使程序文件变得非常大，进而影响页面的响应速度。

在 Flex 3.0 中对一些常用的数据验证功能进行了封装，提供了一组数据验证组件。数据验证组件是专门针对数据验证的特殊组件，包含一定规则的验证及出错提示。使用数据验证组件使得对数据验证更加方便，摆脱了复杂的验证逻辑，有利于应用程序的开发。

Flex 3.0 中提供了一些常用的组件进行数据验证，这些组件能满足用户的基本要求。若用户想自定义所需的数据验证，可继承类型相近的验证组件，从而创建自定义的数据验证组件。Flex 3.0 提供的数据验证组件，包括 CreditCardValidator、CurrencyValidator、DateValidator 等。这些组件可以有效地完成验证任务，出错提示也很丰富。

Flex 3.0 提供的基本数据验证组件如表 14-1 所示。

表 14-1　基本数据验证组件

组件名	说明
CreditCardValidator	信用卡号码验证
CurrencyValidator	货币验证
DateValidator	日期验证
E-mailValidator	E-mail 验证
NumberValidator	数字验证
PhoneNumberValidator	电话号码验证
RegExpValidator	正则表达式验证
SocialSecurityValidator	美国 SocialSecurity 号码验证
StringValidator	字符串验证
ZipCodeValidator	邮编验证

Flex 3.0 提供的一些数据验证组件是以美国的规则制定的。例如，美国邮编长度为 5 位，中国的邮编长度为 6 位。当验证组件不能满足要求时，用户需要修改组件的某些属性从而达到要求。例如，数字验证组件中可指明最大值和最小值。若用户有特殊的验证要求而组件未提供属性或方法时，用户可用自定义组件来实现验证效果。

14.4.2　使用数据验证组件

Flex 3.0 中，所有的数据验证组件都存放在 mx.Validators.Validator 下，并且所有的数据验证组件都是非可见组件，并不显示在 Components 面板中。用户可以通过 MXML 标签形式和在程序中创建数据验证组件对象两种方式来使用数据验证组件。

1. 使用数据验证组件

使用数据验证组件的语法如下所示。

```
<mx:验证组件 source="{需验证的数据组件 id}"property="数据组件的属性">
```

其中，source 属性表示要验证的对象，property 属性表示验证对象的属性。source 属性和 property 属性是数据验证组件最基本的两个属性。例如，邮箱验证组件对某文本框的 text 属性进行验证。

例如，使用<mx:PhoneNumberValidator>组件对一个文本框中的文本进行电话号码验证，具体代码如代码 14.32 所示。

代码 14.32　<mx:PhoneNumberValidator>组件验证电话号码

```
<xml version="1.0"?>
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<!--定义"PhoneNumberValidator"，用以校验电话号码-->
<mx:PhoneNumberValidator id="checkphone" source="{phoneInput}" property=
"text"/>
<mx:Panel>
<mx:TextInputid="phoneInput"/>
<mx:TextInput id="zipCodeInput"/>
</mx:Panel>
</mx:Application>
```



由于验证组件都为不可视化组件，所以不能放置于可视化组件内。在代码 14.32 中<mx:PhoneNumberValidator>组件就不能放置于<mx:Panel>组件中。

2．在程序中创建数据验证对象

在 Flex 中，除了使用 MXML 标签形式创建数据验证对象外，还可以在程序代码中创建数据验证对象。当然，如果只需要简单验证的话，使用标签方式就已经足够。下面使用编程的方式来创建用于验证电话号码的数据验证对象，具体如代码 14.33 所示。

代码 14.33　编程方式验证电话号码

```
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
<mx:Script>
[CDATA[
//添加引用
import mx.validators.PhoneNumberValidator;
//创建电话号码验证对象
private var v:PhoneNumberValidator = new PhoneNumberValidator();
private function createValidator():void{
//设置验证对象属性
v.source = phoneInput;
v.property = "text";
}
]]>
mx:Script>
<mx:Panel>
<mx:TextInput id="phoneInput" creationComplete="createValidator();"/>
<mx:TextInput id="zipCodeInput"/>
</mx:Panel>
mx:Application>
```

在使用编程的方式创建数据验证对象时，首先需要添加对该对象的引用，然后就可以使用该对象的构造方法创建一个新的对象实例，并且设置对象的 source 属性、property 属性以及

其他属性等。

14.4.3 验证触发方式

验证触发方式是指用户采用何种动作触发验证。常用的触发方式有默认触发和任意动作触发。默认触发是指当焦点离开输入源时触发验证。任意动作触发是指用户可指定某一动作触发验证。前者是开发过程中最常使用到的一种方式，后者比较灵活，也比较容易理解。

1. 默认触发验证

默认触发验证是当用户把焦点离开输入源时触发默认触发验证。其语法如下所示。

```
<mx:验证组件类型 source="{输入源 id}" property="输入源的属性" />
```

默认触发方式只需确定验证组件的 source 属性和 property 属性，其他属性默认即可。下面创建一个简单的实例，采用默认触发验证方式，当焦点离开文本框时触发验证。具体如代码 14.34 所示。

代码 14.34　验证用户名

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        import mx.controls.Alert;
    </mx:Script>
    <mx:StringValidator source="{fname}" property="text" minLength="4"
    maxLength="20"/>
    <mx:Panel title="StringValidator Example">
            <mx:Form>
                 <mx:FormItem label="请输入用户名:(4-20 位字符) ">
                   <mx:TextInput id="fname" width="100%"/>
               </mx:FormItem>
               <mx:FormItem >
                   <mx:Button id="myButton" label="验证用户名" />
               </mx:FormItem>
            </mx:Form>
    </mx:Panel>
</mx:Application>
```

在代码 14.34 中，创建了一个 StringValidator 对象，用于验证用户输入的用户名，设置其 source 属性和 property 属性，即验证对象为用户的输入值，并且设置其 minLength 和 maxLength 属性分别为 4 和 20，即用户名长度必须大于 4 个字符并且不得大于 20 个字符。当焦点离开文本框时，就会触发验证。

验证组件的结果直接反映在对象组件上，本程序中若文本框组件 fname 验证错误，结果为文本框边框颜色变红色，鼠标放置到文本框上，将会显示具体的错误提示信息。具体效果

如图 14-20 所示。验证错误的提示信息存储于数据组件的 errorString 属性中。

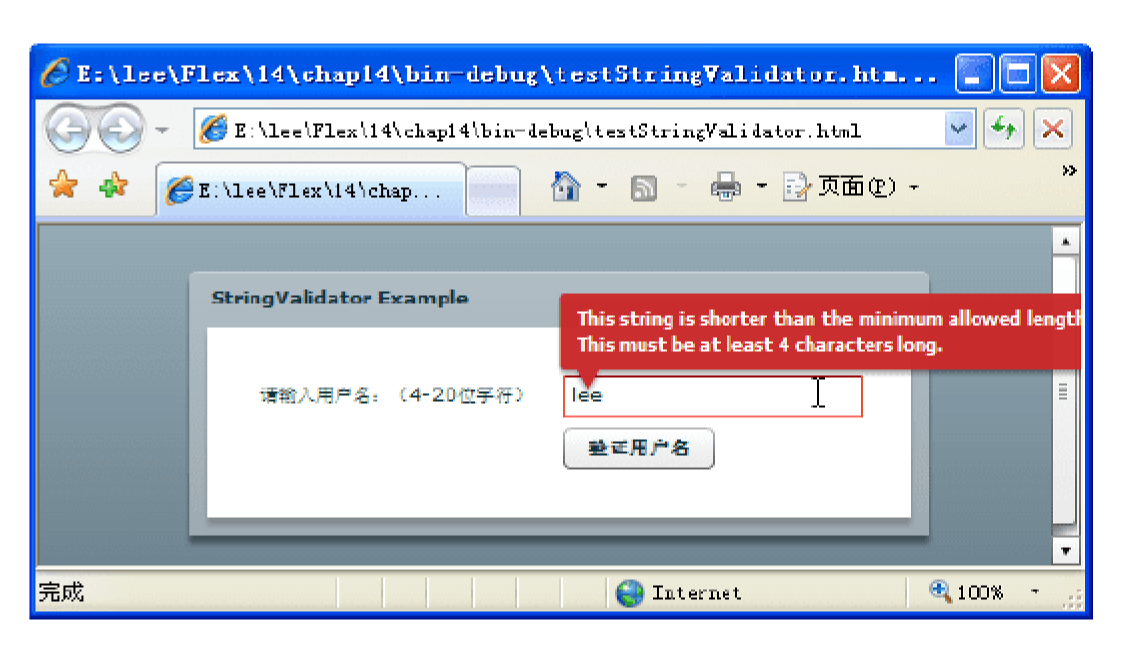


图 14-20　验证用户名

2. 任意动作触发验证

默认触发验证虽然实现了验证，但感觉验证的功能还是可有可无，当用户把鼠标移到文本框上再移出才会触发验证。如果用户预先不把鼠标移到文本框上，这个验证就失去了效果。这时就可以根据需要触发验证。例如，利用【提交】按钮检查验证是否通过，如果验证未通过则文本框显示为红色。

任意动作触发验证有两种写法：一种是在验证组件中指明触发器对象和触发动作，另一种是执行事件处理函数。

在验证组件中指明触发器和触发动作的语法如下所示。

```
<mx:验证组件类型
source="{输入源 id}"
property="输入源的属性"
trigger="{触发器对象}"
triggerEvent="触发事件"
>
```

trigger 属性指明触发验证的组件，也称为触发器。triggerEvent 属性表示触发验证组件的方法。

下面创建一个简单的实例，使用 Button 组件的 click 事件触发验证，具体如代码 14.35 所示。

代码 14.35　click 事件触发验证

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        import mx.controls.Alert;
    </mx:Script>
   <mx:EmailValidator source="{email}" property="text"
       trigger="{myButton}" triggerEvent="click"
       valid="Alert.show('邮箱验证成功! ');"/>
```

```
    <mx:Panel title="EmailValidator Example" width="75%" height="75%"
        paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom="10">
        <mx:Form>
            <mx:FormItem label="请输入你的电子邮箱: ">
                <mx:TextInput id="email" width="100%"/>
            </mx:FormItem>
            <mx:FormItem >
                <mx:Button id="myButton" label="验证邮箱" />
            </mx:FormItem>
        </mx:Form>
    </mx:Panel>
</mx:Application>
```

代码 14.35 中创建了一个 EmailValidator 组件，对用户输入的电子邮箱格式进行验证，设置 trigger 属性值为组件 myButton，并且设置 triggerEvent="click"，即当用户单击按钮时触发验证。具体效果如图 14-21 所示。

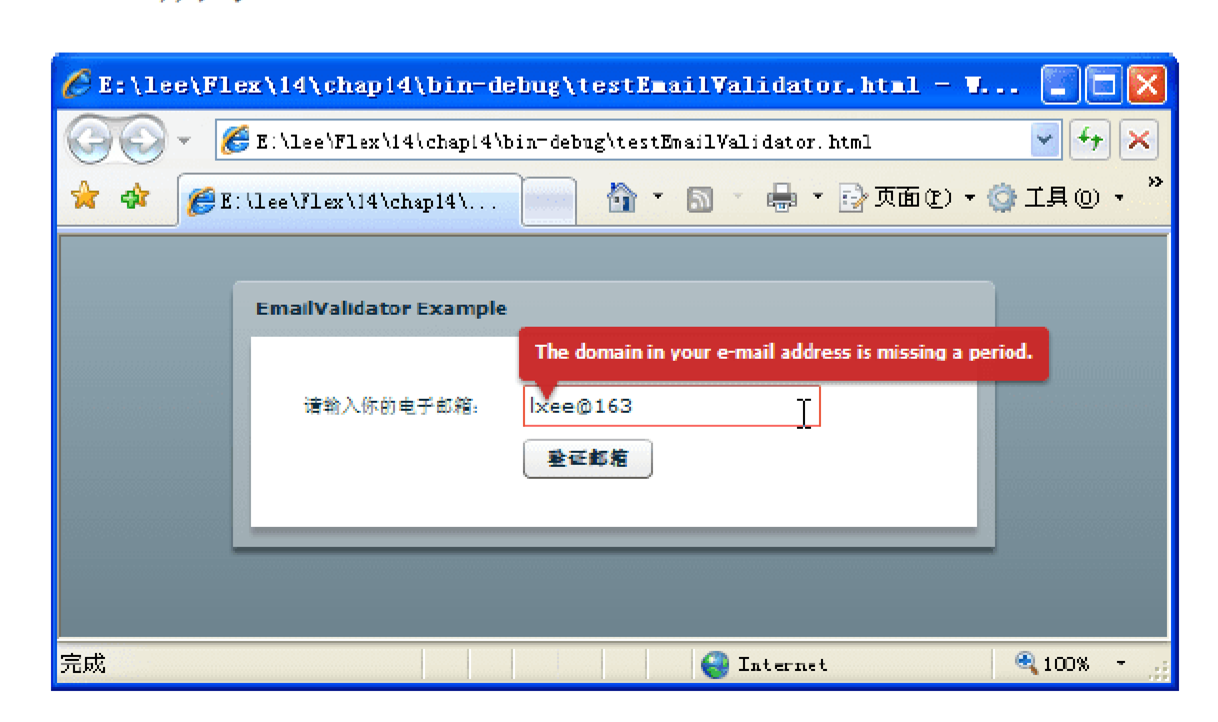


图 14-21　验证电子邮箱

任意动作的触发验证也可采用代码触发验证方式。代码触发验证的方式符合 Flex 3.0 的事件机制，更容易理解。其语法如下所示。

```
<组件事件="验证组件.validate();"/>
```

验证组件都包含一个 validate()方法，用以代码方式执行验证。上述实例使用代码方式触发验证的代码如代码 14.36 所示。

代码 14.36　代码方式执行验证

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        import mx.controls.Alert;
    </mx:Script>
    <mx:EmailValidator id="checkEmail" source="{email}" property="text" valid=
    "Alert.show('邮箱验证成功! ');"/>
    <mx:Panel title="EmailValidator Example" width="75%" height="75%"
```

```
        paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom="10">
        <mx:Form>
            <mx:FormItem label="请输入你的电子邮箱: ">
                <mx:TextInput id="email" width="100%"/>
            </mx:FormItem>
            <mx:FormItem >
                <mx:Button id="myButton" label="验证邮箱" click="checkEmail.
                validate()"/>
            </mx:FormItem>
        </mx:Form>
    </mx:Panel>
</mx:Application>
```

14.4.4　验证失败处理

数据验证的目的就是为了阻止用户输入格式错误的数据，那么当验证失败时就需要提示具体的错误信息。验证组件中提供了丰富的错误类型，只是这些错误类型的提示是英文的，用户可能需要改变提示。修改错误提示的方法是修改组件中相应的错误类型属性。例如，PhoneNumberValidator 组件中的 wrongLengthError 属性表示长度错误提示。

用户可根据需要修改相应的出错信息。其语法如下所示：

```
<验证组件错误类型属性="自定义错误提示"/>
```

例如，下面创建了一个 EmailValidator 对象，对用户输入的电子邮箱进行验证，并且还定义了具体的错误提示信息，具体代码如代码 14.37 所示。

代码 14.37　验证时显示错误提示信息

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"horizontal" fontSize="13">
<mx:Script>
<![CDATA[
import mx.events.ValidationResultEvent;//引用 ValidationResultEvent 类
import mx.controls.Alert;//引用 Alert 类
private function checkHandle():void//验证处理函数
{
if(emailV.validate().type==ValidationResultEvent.VALID)//验证通过
{
Alert.show("电子邮件验证成功");//提示"验证成功"
}
}
]]>
</mx:Script>
<!--电子邮箱验证组件-->
```

```
<mx:EmailValidator
id="emailV"
source="{txtEmail}"
property="text"
invalidCharError="非法字符"
invalidDomainError="非法域"
invalidIPDomainError="非法 IP 域"
missingAtSignError="缺少@符"
missingPeriodInDomainError="缺少域后缀"
missingUsernameError="缺少用户名"
/>
<mx:Panel title="EmailValidator 验证失败处理" width="352" height="171"
horizontalAlign="center" verticalAlign="middle" fontSize="12">
<mx:TextInput id="txtEmail" fontSize="10"/><!--输入框组件-->
<!--Label 组件，用于显示验证结果-->
<mx:Label text="{txtEmail.errorString}" color="#E70F38"/>
<mx:Button id="mySubmit" label="验证邮箱" click="checkHandle();" fontSize=
"10"/><!--按钮组件，用于验证处理-->
</mx:Panel>
</mx:Application>
```

在代码 14.37 中，首先创建电子邮件验证组件 EmailValidator 并设置其验证对象。接下来，自定义设置具体的错误提示信息，Flex 3.0 为每个数据验证组件都提供丰富的验证错误类型。用户可以通过为不同的错误类型分别设置具体的提示信息，来实现精确提示的效果。在本实例中，分别设置了 invalidCharError、invalidDomainError、invalidIPDomainError、missingAtSignError、missingPeriodInDomainError、missingUsernameError 等错误类型的具体提示信息。具体的效果如图 14-22、图 14-23 所示。



图 14-22　缺少用户名

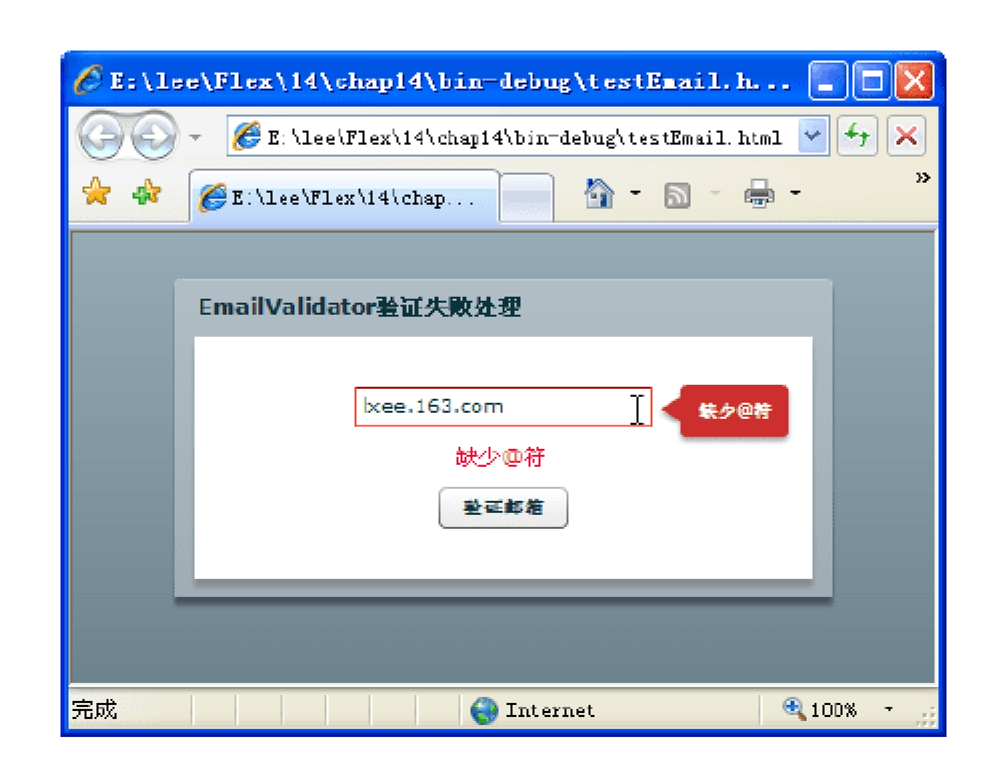


图 14-23　缺少@符号

设置【验证邮箱】按钮的 click 事件调用函数 checkHandle()，在该函数中，使用 if 语句判断是否验证成功。

```
if(emailV.validate().type==ValidationResultEvent.VALID)
```

ValidationResultEvent 类包含于 mx.events.*中，是验证结果事件类。其中，INVALID 值表

示验证失败，VALID 值表示验证成功。验证成功效果如图 14-24 所示。

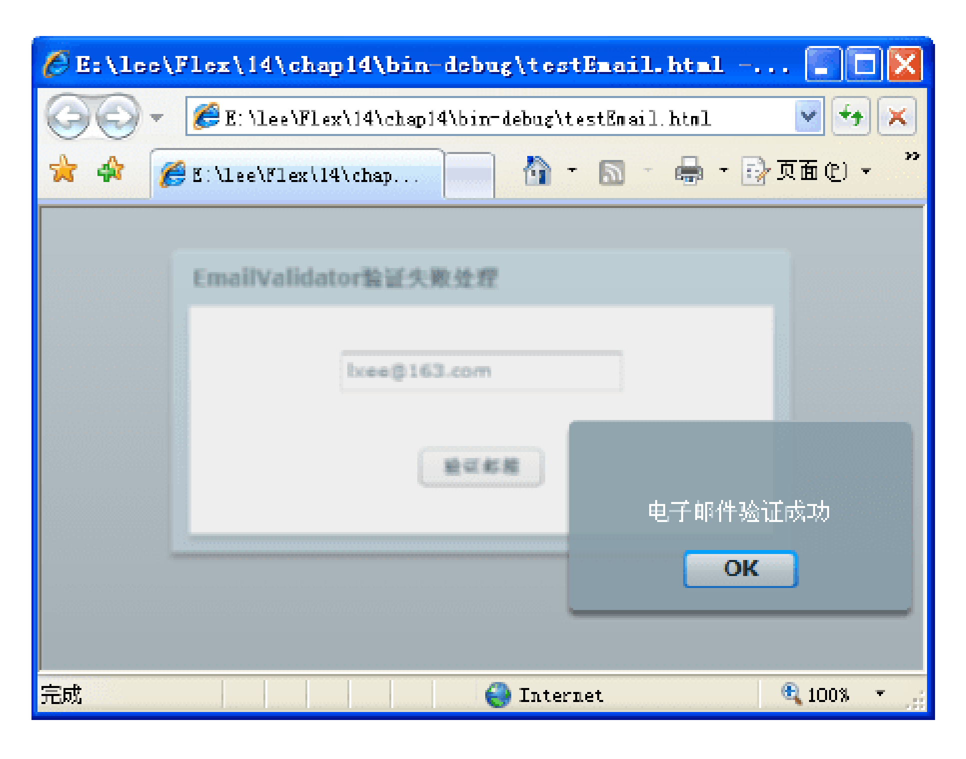


图 14-24　邮箱验证成功

14.4.5　自定义验证组件

虽然 Flex 3.0 对表单中经常遇到的数据验证功能进行了封装，创建了一系列的数据验证组件。但是对于用户来说，如果需要实现一些特殊的验证请求，这些验证组件就无法满足要求了。在 Flex 3.0 中，当验证组件不能满足用户的特殊验证需求时，用户可考虑创建自定义验证组件。一般来说，多条件复杂数据验证都需要自定义验证组件。例如，字符串长度为 3~43，内容不能包含某些特殊字符。

Flex 3.0 中自定义验证组件需要首先继承功能最相近的验证组件，然后重写验证组件中的 doValidation()方法。创建自定义验证组件的步骤如下所示。

（1）选择 File | New| ActionScript Class 命令，弹出 New ActionScript Class 对话框。

（2）在 Package 文本框中输入相对路径，在 Name 文本框中输入类名，在 Superclass 文本框中输入继承类，此处为 mx.validators.validator。启用 Generrate constructor from superclass 复选框表示使用继承类的构造函数。单击 Finish 按钮，完成类的创建，如图 14-25 所示。

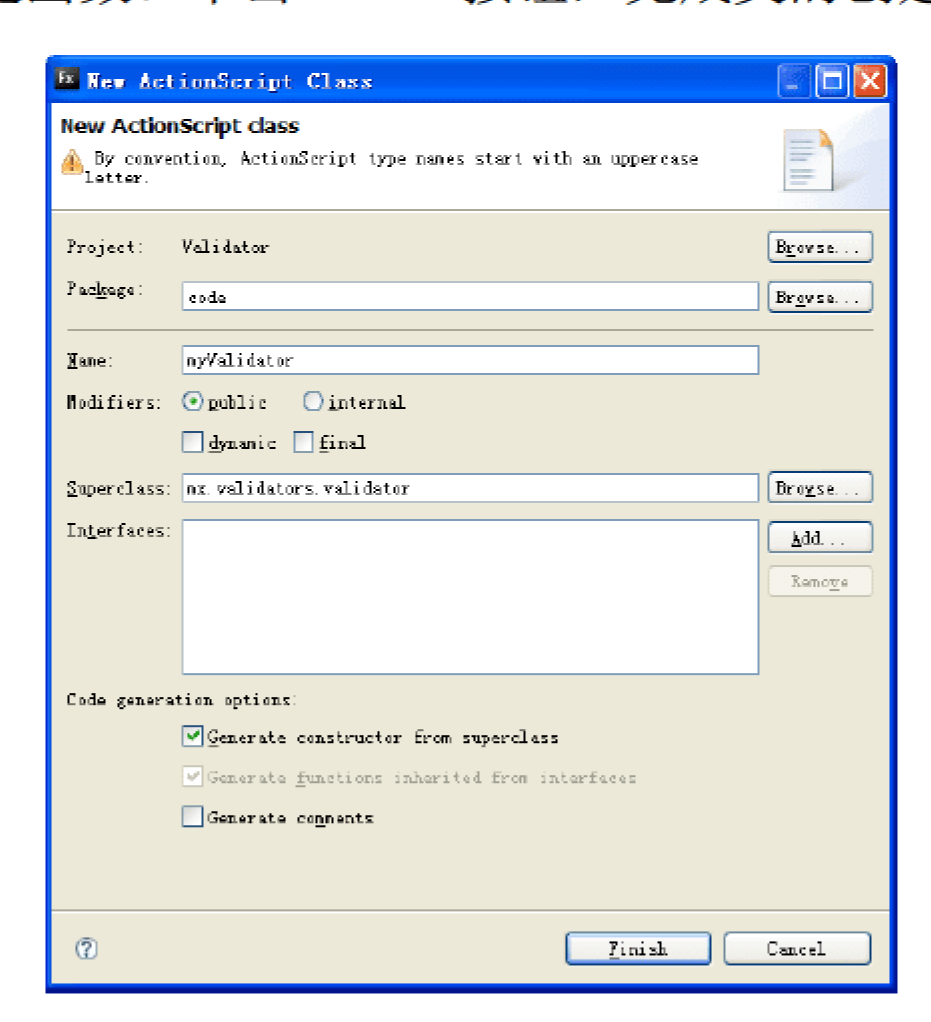


图 14-25　创建自定义验证组件

（3）编写自定义验证组件类 myValidators.as，具体代码如代码 14.38 所示。

代码 14.38　myValidators.as 类

```
//myValidator.as
package code
{
    import mx.validators.Validator;//引用 Validator 类
    import mx.validators.ValidationResult;//引用 ValidationResult 类
    public class myValidator extends mx.validators.validator
    {
        public function myValidator()//构造函数
        {
            super();
        }
       private var results:Array;//定义一个数组，用以存储错误
       //重写验证函数
       override protected function doValidation(value:Object):Array
       {
       var s:String=value as String;
       results=[];//清空数组
       results=super.doValidation(value);//先用继承类中的 doValidation()方法验证
       if(results.length>0)//如果验证时有错，返回错误信息
       return results;
       if(s.length>10)//自定义验证，字符长度不超过 10
       {
        //记录出错信息
        results.push(new ValidationResult(true,"text","StringTooLong","密码
        长度不能超过 10"));
       }
       if(s.length<4)//自定义验证，字符长度不低于 4
       {
        //记录出错信息
        results.push(new ValidationResult(true,"text","StringTooShort","密码
        长度不能低于 4"));
       }
       return results;
       }
    }
}
```

在代码 14.38 中，创建了一个用户自定义验证组件类 myValidator，该类继承自 mx.validators.validator 基类。使用 override 关键字重写函数 doValidation()。在该函数中，首先使用 super.doValidation(value)语句执行继承类中的 doValidator()方法，然后自定义具体的验证需求，并将结果存放在数组 results 中。

ValidationResult 类是验证结果类，包含于 mx.validators.*中。ValidationResult 类的构造函数有 4 个参数。第 1 个参数表示是否为错误类型，第 2 个参数表示指向某特定属性，第 3 个参数表示错误类型，第 4 个参数表示错误提示。本程序中自定义了 StringTooLong 和 StringTooShort 错误类型，分别提示“密码长度不能超过 10”和“密码长度不能低于 4”，此错误在数据组件的长度不满足条件时发生。

（4）调用自定义验证类。调用自定义组件时必须指明命名空间。引用自定义组件的语法如下所示。

```
<最外层组件…xmlns:命名空间="类所属的包">
...
<命名空间:自定义组件/>
```

命名空间可任意取名。类所属的包必须与类定义时的 package 关键字相同。具体引用自定义组件 myValidator 的代码如代码 14.39 所示。

代码 14.39　代码方式执行验证

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns:Mycode=
"code.*" fontSize="13">
<!--自定义组件 myValidator，用于验证密码-->
<Mycode:myValidator id="myVal"
source="{pwd}" property="text"
/>
<mx:Panel title="自定义验证组件" width="448" height="227" horizontalAlign=
"center" verticalAlign="middle" fontSize="12">
 <mx:Form>
     <mx:FormItem label="请输入用户名:  ">
      <mx:TextInput id="fname" width="100%"/>
    </mx:FormItem>
    <mx:FormItem label="密码（4-10 位字符）: " >
      <mx:TextInput id="pwd" width="100%" displayAsPassword="true"/>
    </mx:FormItem>
    <mx:FormItem >
    <mx:Button id="myButton" label="提交" click="myVal.validate();" />
    </mx:FormItem>
</mx:Form>
 <mx:Label text="{pwd.errorString}" color="#F21B2F"/>
</mx:Panel>
</mx:Application>
```

在代码 14.39 中，引用自定义组件 myValidator，用于验证用户输入的密码是否符合密码策略。具体效果如图 14-26、图 14-27、图 14-28 所示。其中，图 14-26 是继承 Flex 中数据验证组件的错误类型，而图 14-27、图 14-28 则是用户自定义的错误类型。

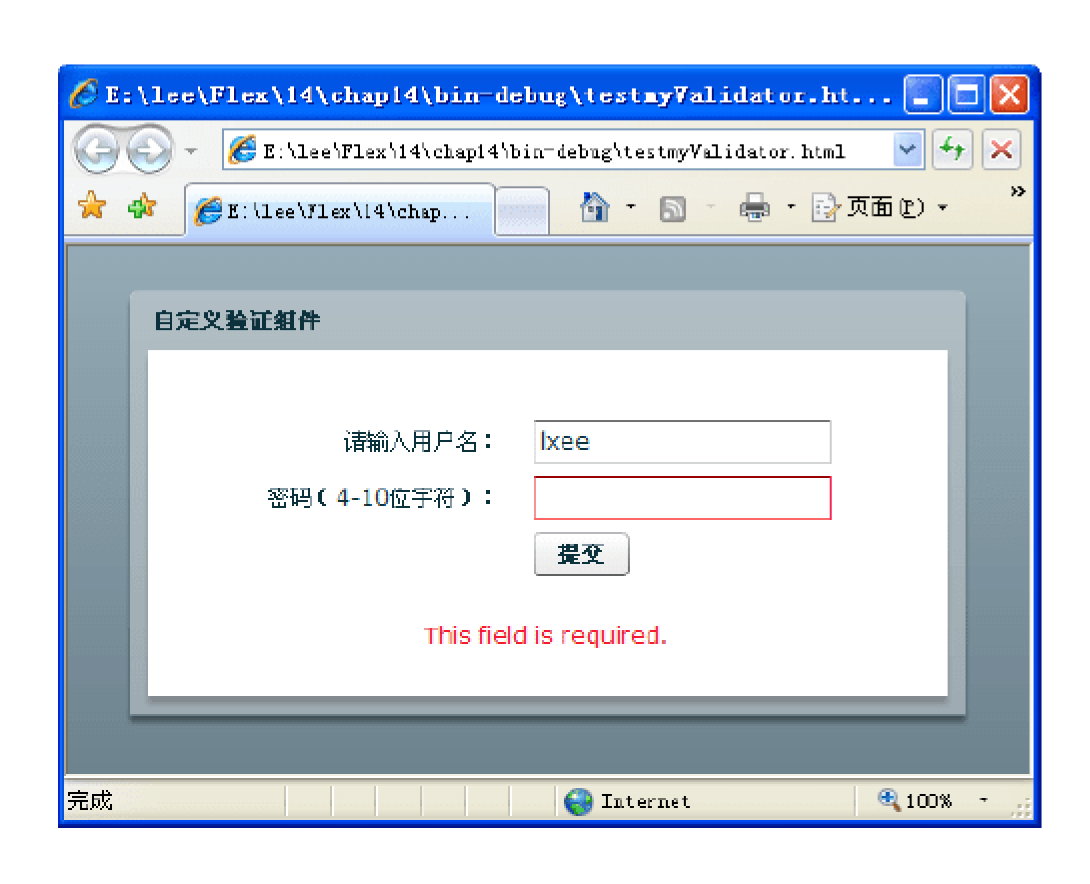


图 14-26　密码不能为空

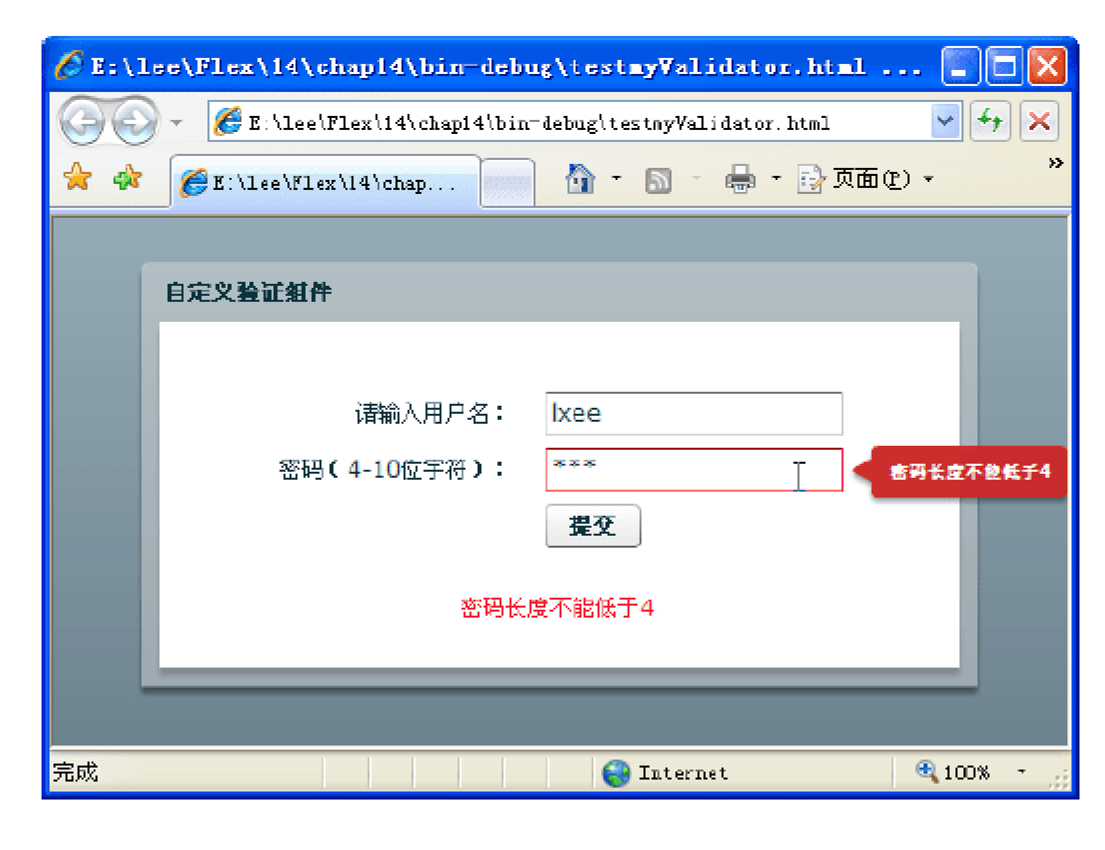


图 14-27　密码长度不能低于 4

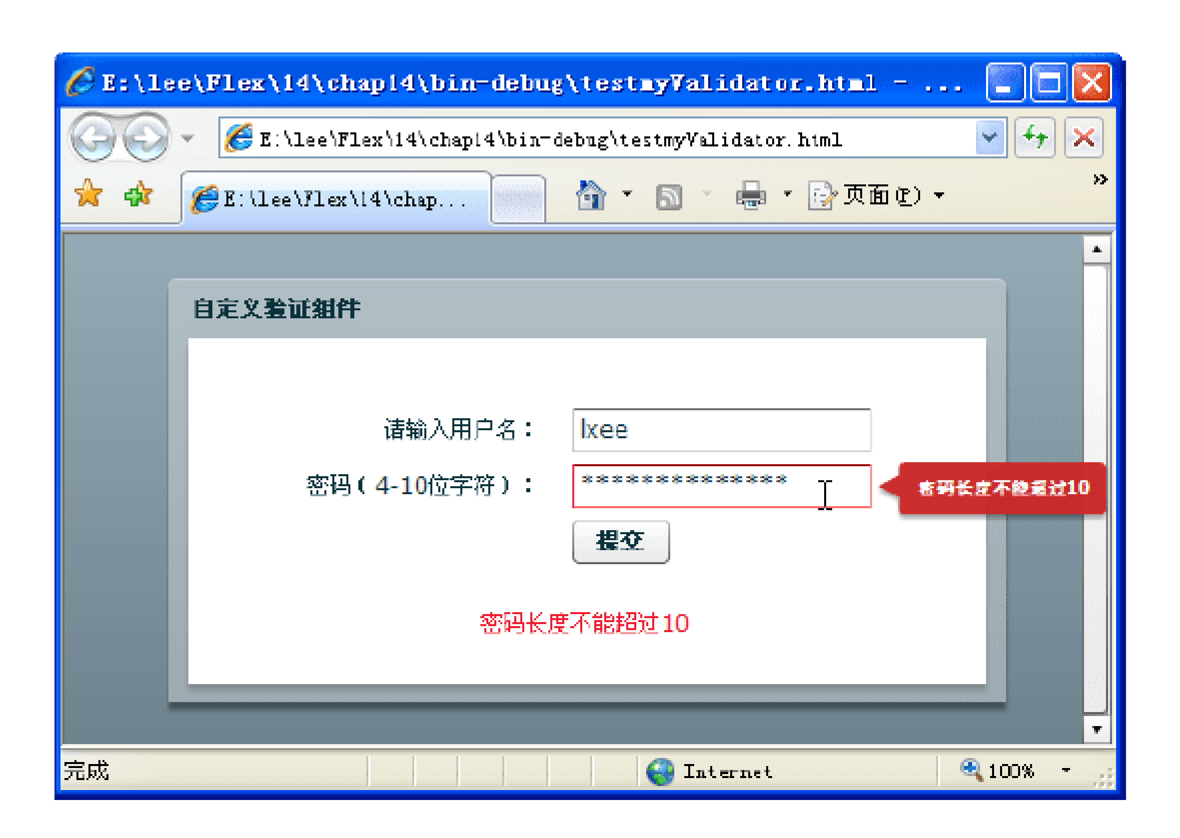


图 14-28　密码长度不能超过 10

14.4.6　数据验证应用实例

用户注册模块是一个网站最基本的模块之一。在用户注册模块，需要把用户提交的注册信息存储到数据库中。为了确保用户注册信息的正确性以及满足数据库中特定的数据类型，在存储注册信息之前，就需要对用户输入的用户名、密码、电子邮件、电话号码等信息进行数据验证。

在本节中，使用前面介绍过的数据验证组件，创建一个简单的验证用户注册信息实例，具体步骤如下。

（1）首先设计用户注册模块的界面。在用户注册模块，需要用户提交用户名、密码、性别、电子邮件、手机号码等信息。在本实例中，使用 Panel 组件和 Form 组件进行布局，并在 Form 组件中使用各种基本组件，如 TextInput、RadioButton 等，接受用户的注册信息。界面的具体设计如代码 14.40 所示。

代码 14.40　用户注册

```
<mx:Panel x="34" y="21" width="428" height="350" layout="absolute" title=
```

```
"用户注册" fontSize="12" borderColor="#D89FF3">
        <mx:Form x="42.5" y="20" width="322" height="237">
            <mx:FormHeading label="填写你的注册信息" width="277"/>
            <mx:FormItem label="用户名: ">
                <mx:TextInput id="username"  fontSize="10" width="174"/>
            </mx:FormItem>
            <mx:FormItem label="密码: ">
            <mx:TextInput id="pwd" displayAsPassword="true"  fontSize="10"
            width="174"/>
            </mx:FormItem>
            <mx:FormItem label="确认密码: ">
            <mx:TextInput id="repwd" displayAsPassword="true"  fontSize="10"
            width="173"/>
            </mx:FormItem>
            <mx:FormItem label="性别: ">
                <mx:HBox>
                <mx:RadioButton id="man" label="男" groupName="sexgroup"
                selected="true"/>
                <mx:RadioButton id="woman" label="女" groupName="sexgroup"/>
                </mx:HBox>
            </mx:FormItem>
            <mx:FormItem label="电子邮箱: " width="254">
                <mx:TextInput id="email" fontSize="10" width="174"/>
            </mx:FormItem>
            <mx:FormItem label="手机号码: ">
                <mx:TextInput id="phone" fontSize="10" width="173"/>
            </mx:FormItem>
        </mx:Form>
        <mx:Button label="取消" x="207" y="265"/>
        <mx:Button label="提交" x="108" y="265" click="submit()"/>
</mx:Panel>
```

（2）界面设计完成后，就可以根据需求创建数据验证组件，对用户输入的注册信息进行验证。在本实例中，分别对用户名、电子邮件、电话号码等信息进行验证，具体如代码 14.41 所示。

代码 14.41　对数据进行验证

```
<mx:PhoneNumberValidator id="pval" allowedFormatChars="" source="{phone}"
property="text" invalidCharError="非法字符，请检查"/>
<mx: id="sval" minLength="6" maxLength="12" source="{username}" property=
"text" tooLongError="用户名长度不得超过 12 位" tooShortError="用户名不得低于 6 位"/>
<mx:EmailValidator id="eval" source="{email}" property="text" invalidChar-
Error="非法字符" invalidDomainError="非法域" invalidIPDomainError="非法 IP 域"
missingAtSignError="缺少@符" missingPeriodInDomainError="缺少域后缀"
missingUsernameError="缺少用户名"/>
```

在代码 14.41 中，创建了 3 个数据验证对象：PhoneNumberValidator、StringValidator、EmailValidator，分别设置这 3 个数据验证组件的 source、property 属性，并且对可能出现的错误类型设置具体的错误提示信息。具体的验证效果如图 14-29 所示。

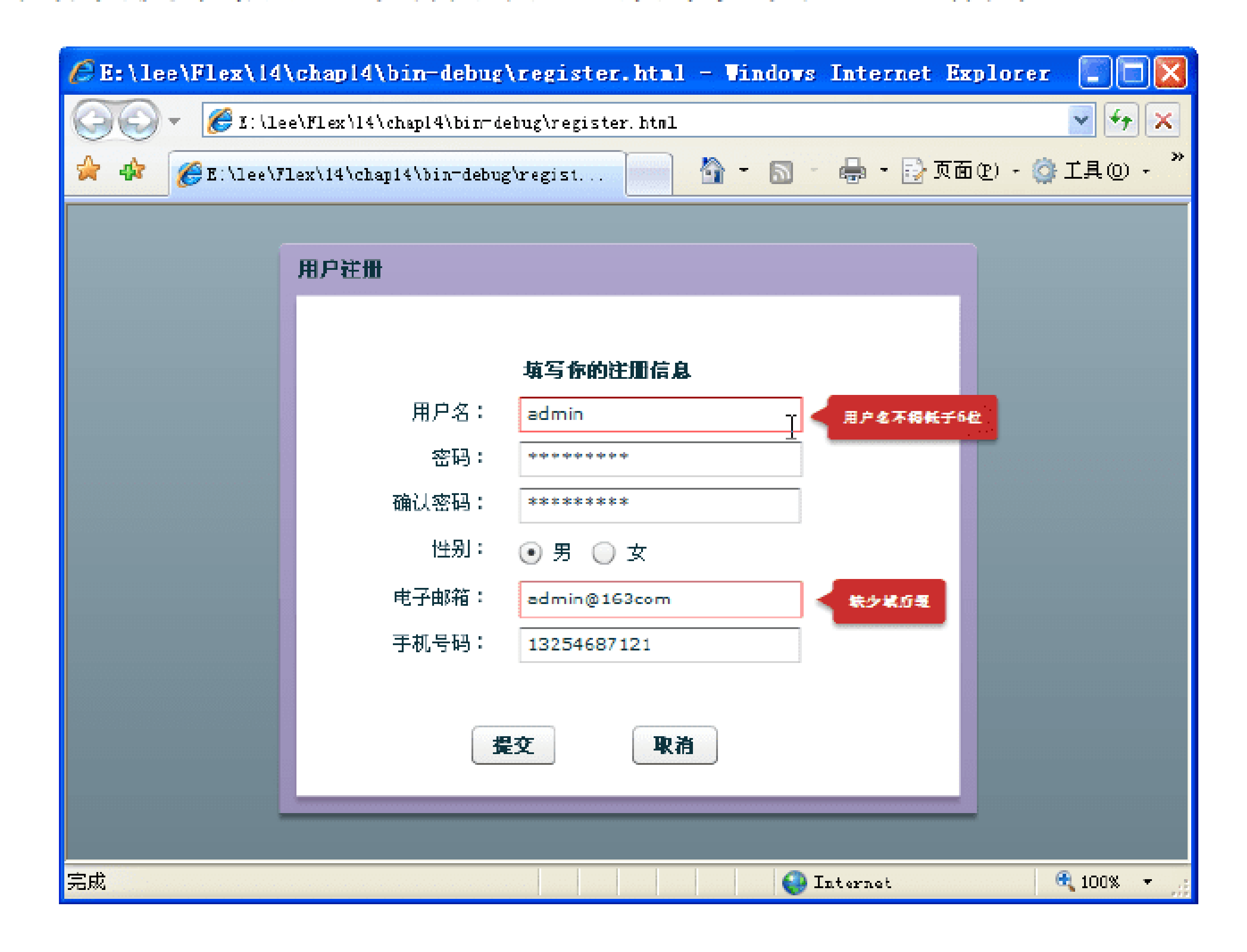


图 14-29　数据验证错误提示效果

（3）在上面创建了 3 个数据验证组件，当焦点离开文本框时，就会触发相应的数据验证组件。但如果用户并不将焦点移入到文本框，直接单击【提交】按钮，就不会触发相应的验证组件。这时就需要在提交数据的时候对所有验证组件的验证结果进行判断，当所有验证组件都通过验证后，才可以提交数据。在本实例中，创建一个函数 submit()，当用户单击【提交】按钮时被调用，具体如代码 14.42 所示。

代码 14.42　“提交”函数 submit()

```
<mx:Script>
    <![CDATA[
    import mx.controls.Alert;
    import mx.events.ValidationResultEvent;
    private function submit():void{
    //若用户名未占用且各种校验都正确
     if(eval.validate().type==ValidationResultEvent.VALID&&sval.validate().
     type==ValidationResultEvent.VALID&&pval.validate().type==Validation-
     ResultEvent.VALID)
     {
    //密码为空的情况时，警告
      if(pwd.text==""||repwd.text=="")
      {
       Alert.show("密码不能为空");}
       //两次输入密码不一致时，警告
      else if(pwd.text!=repwd.text)
       Alert.show("密码不一致");
```

```
        else//注册成功时，提示
         Alert.show("注册成功");
        }
       else
        {
        Alert.show("校验有误，请检查你的输入");
        }
      }
     ]]>
    </mx:Script>
```

在函数 submit()中，首先使用 if 语句判断所有验证组件的验证结果，所有验证组件都验证无误后再判断用户两次输入的密码是否为空和是否一致，符合全部要求后，对用户提交的数据进行存储及相应处理（对数据的存储并不是本章的主要内容，这里就不设计具体的程序代码）。单击【提交】按钮后的验证效果如图 14-30 所示。

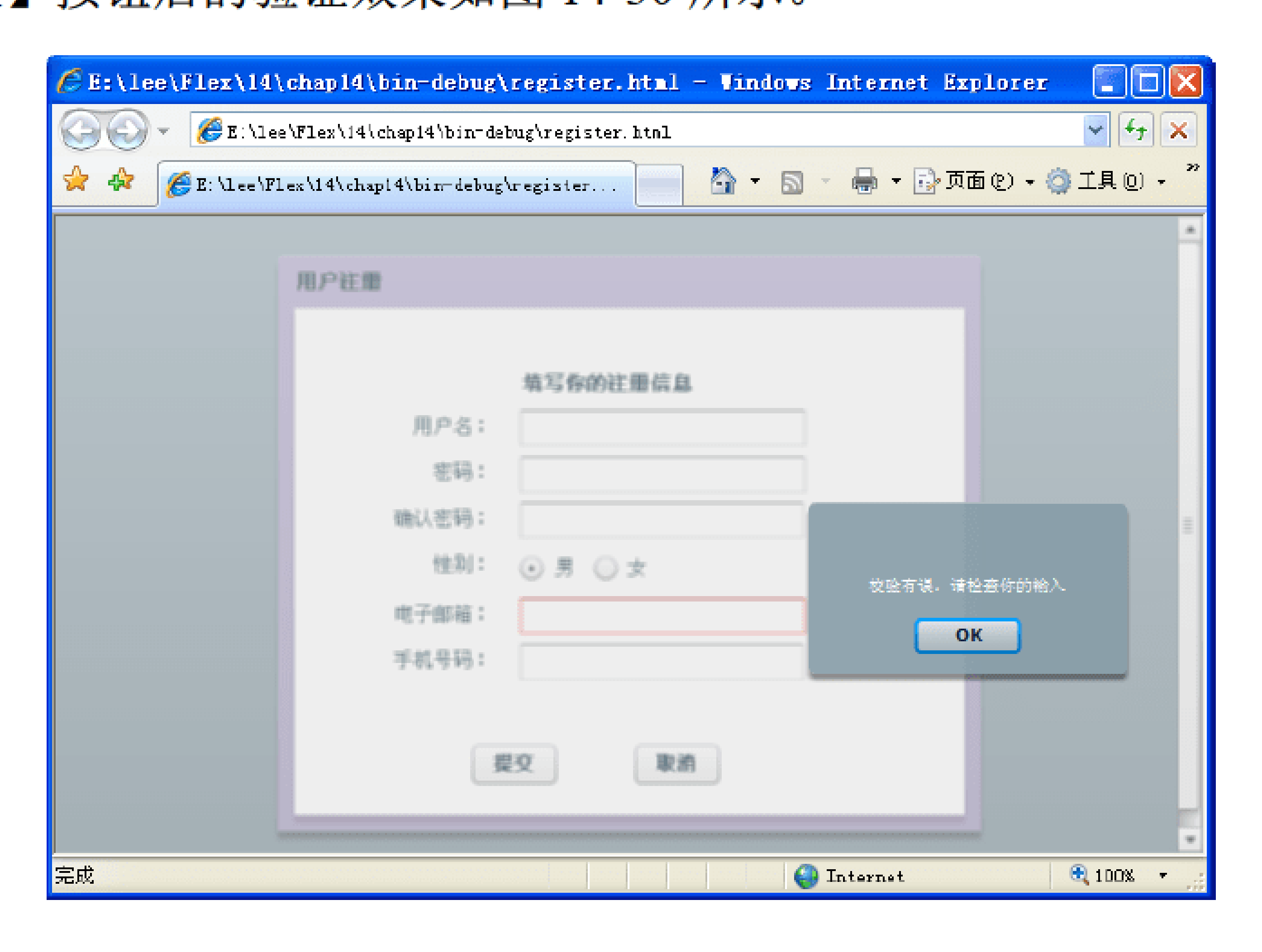


图 14-30　判断验证结果

14.5　数据格式化

数据格式化就是对某些特殊数据的格式进行规范。例如，日期格式有很多种，可以为“2009-09-11”或者“09/11/2009”等。甚至有时对数据进行规范是必须的，像货币的格式要统一。

14.5.1　格式化组件概述

Flex 3.0 中提供了几种常见的数据格式化组件，包括 DateFormatter、NumberFormatter、PhoneFormatter 等，如表 14-2 所示。

表 14-2　Flex 3.0 中的数据格式化组件

组件名	说明	组件名	说明
CurrencyFormatter	对货币数据格式化	PhoneFormatter	对电话号码数据格式化
DateFormatter	对日期数据格式化	ZipCodeFormatter	对邮编数据格式化
NumberFormatter	对数字数据格式化		

在 Flex 中所有的 formatter（格式器）都是 mx.fomatters.Formatter 类的子类。一个 Formatter 类声明一个 format()方法，这个方法需要一个参数值，返回一个字符串。

对于大多数的 formatter，当一个错误发生时，将返回一个空字符串，并且描述错误的信息被写在 formatter 的 error 属性中。error 属性继承自 Formatter 类。

14.5.2　货币格式化组件<mx:CurrencyFormatter>

<mx:CurrencyFormatter>组件用以格式化货币，其常用的属性如表 14-3 所示。

表 14-3　<mx:CurrencyFormatter>组件的常用和属性

属性名	说明
alignSymbol	货币符号位置。其值可为 left 或 right
currencySymbol	货币符号，如$、￥、£
useThousandsSeparator	是否使用千位符（,）。其值可为 true 或 false
useNegativeSign	是否使用负号。其值可为 true 或 false
precision	格式化数据的精度
error	格式化数据出错时的提示信息

下面创建一个使用<mx:CurrencyFormatter>组件格式化货币的实例，具体如代码 14.43 所示。

代码 14.43　<mx:CurrencyFormatter>组件格式化货币

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.events.ValidationResultEvent;
            private var vResult:ValidationResultEvent;
            // Event handler to validate and format input.
            private function Format():void {
                vResult = numVal.validate();
                if (vResult.type==ValidationResultEvent.VALID){
                    var temp:Number=Number(priceUS.text);
                    formattedUSPrice.text=rmbFormatter.format(temp);
                 }
                 else {
                    formattedUSPrice.text="";
                 }
```

```
                }
            ]]>
        </mx:Script>
        <mx:CurrencyFormatter id="rmbFormatter" precision="2"
            currencySymbol="￥" decimalSeparatorFrom="."
            decimalSeparatorTo="." useNegativeSign="true"
            useThousandsSeparator="true" alignSymbol="left"/>
        <mx:NumberValidator id="numVal" source="{priceUS}" property="text"
            allowNegative="true" domain="real"/>
        <mx:Panel title="货币格式化组件示例" width="75%" height="75%" fontSize="12"
        horizontalAlign="center" verticalAlign="top" layout="horizontal">
            <mx:Form>
                <mx:FormItem label="输入人民币金额：">
                    <mx:TextInput id="priceUS" text="" width="145" fontSize="10"/>
                </mx:FormItem>
                <mx:FormItem label="格式化后金额：">
                    <mx:TextInput id="formattedUSPrice" text="" width="145"
                    editable="false" fontSize="10" color="#E50B39"/>
                </mx:FormItem>
                <mx:FormItem>
                    <mx:Button label="验证并格式化" click="Format();"/>
                </mx:FormItem>
            </mx:Form>
        </mx:Panel>
    </mx:Application>
```

在代码 14.43 中，创建了一个货币格式化组件<mx:CurrencyFormatter>，设置其 currencySymbol 属性为￥，useNegativeSign 属性为 true，useThousandsSeparator 属性为 true，alignSymbol 属性为 left，并设置其 precision 属性为 2，即精确到小数点后两位。

当用户单击【验证并格式化】按钮时，首先判断用户输入的数据格式是否正确，如果无误则使用 CurrencyFormatter 对象的 format()方法进行格式化，获取格式化后字符串，并在文本框中显示出来。具体的效果如图 14-31 所示。

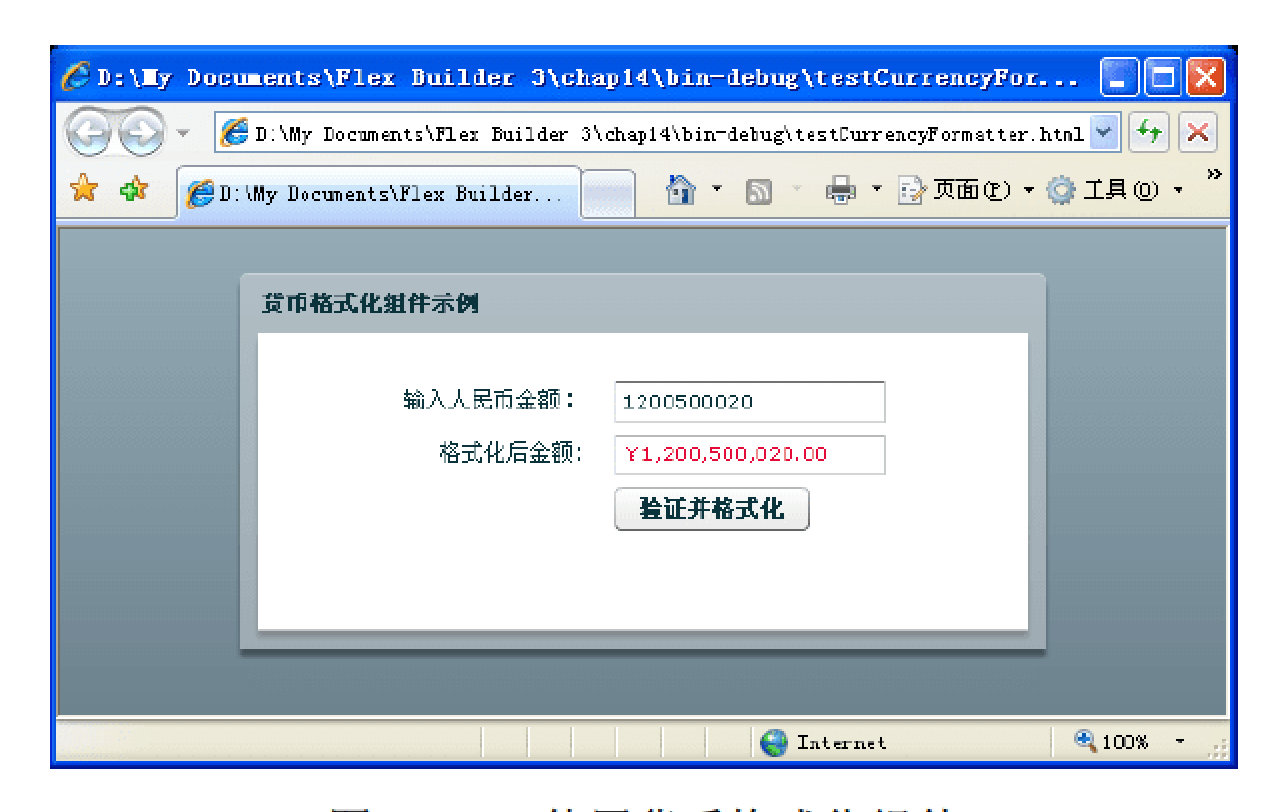


图 14-31　使用货币格式化组件

14.5.3　日期格式化组件<mx:DateFormatter>

<mx:DateFormatter>组件用以格式化日期，其常用的属性如表 14-4 所示。

表 14-4　<mx:DateFormatter>组件的常用属性

属性名	说明
error	格式化数据出错时的提示信息
formatString	格式化掩码

<mx:DateFormatter>组件的 formatString 属性中定义格式化掩码，可用“Y|M|D|A|E|H|J|K|L|N|S”组合生成。日期掩码字符的说明如表 14-5 所示。

表 14-5　日期掩码字符的说明

掩码字符	说明
Y	年份。可用若干个 Y 组成。例如：YY=09，YYYY=2009，YYYYY=02009
M	月份。可用若干个 M 组成。例如：M=7，MM=07，MMM=Jul，MMMM=July
D	天。可用若干个 D 组成。例如：D=4，DD=04
A	am 或 pm
E	星期几。可用若干个 E 组成。例如：E=1，EE=01，EEE=Mon，EEEE=Monday
H	从 1 开始记数的 24 小时制（1-24）
J	从 0 开始记数的 24 小时制（0-23）
K	从 0 开始记数的 12 小时制（0-11）
L	从 1 开始记数的 12 小时制（1-12）
N	分钟。可用若干个 N 组成。例如：N=3，NN=03
S	秒。例如：SS=30

根据表格 14-5 中的掩码字符可组成丰富的日期格式。例如，掩码“EEEE,MMM.D, YYYY‘at’H:NNA”应用的结果为“Tuesday,Sept.8,2009at1:26PM”。

下面使用<mx:DateFormatter>创建一个格式化日期数据的简单实例，具体如代码 14.44 所示：

代码 14.44　<mx:DateFormatter>组件格式化日期

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.events.ValidationResultEvent;
            private var vResult:ValidationResultEvent;
            private function Format():void{
                vResult = dateVal.validate();
                if (vResult.type==ValidationResultEvent.VALID){
                    formattedDate.text=dateFormatter.format(dob.text);
                }
```

```
                else {formattedDate.text= "";}
            }
        ]]>
    </mx:Script>
    <mx:DateFormatter id="dateFormatter" formatString="YYYY年 M月 D日"/>
    <mx:DateValidator id="dateVal" source="{dob}" property="text" inputFormat=
    "mm/dd/yyyy"/>
    <mx:Panel title="日期格式化组件示例" width="95%" height="95%" fontSize="12">
        <mx:Form width="100%">
            <mx:FormItem label="输入日期(mm/dd/yyyy): " width="100%">
                <mx:TextInput id="dob" text="" width="175"/>
            </mx:FormItem>
            <mx:FormItem label="格式化后日期: " width="100%">
              <mx:TextInput id="formattedDate" text="" editable="false" color=
              "red" width="174"/>
            </mx:FormItem>
            <mx:FormItem>
                <mx:Button label="验证并格式化" click="Format();"/>
            </mx:FormItem>
        </mx:Form>
    </mx:Panel>
</mx:Application>
```

在代码 14.44 中，创建了 DateFormatter 对象，并设置其 formatString 属性为“YYYY 年 M 月 D 日”，当用户单击【验证并格式化】按钮后，首先判断用户输入的日期格式，如果格式正确则调用 DateFormatter 对象的 format()方法，格式化数据，并在文本框中显示出来，具体的显示效果如图 14-32 所示。

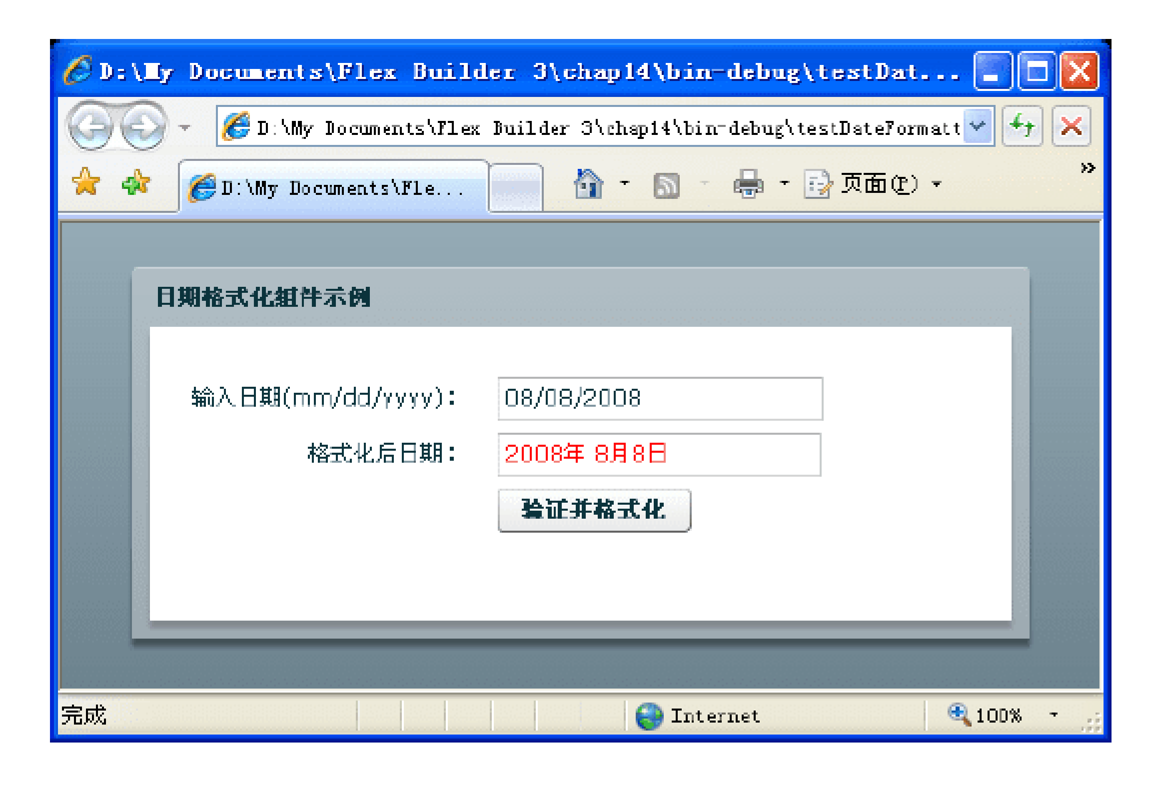


图 14-32　使用日期格式化组件

14.5.4　数字格式化组件<mx:NumberFormatter>

<mx:NumberFormatter>组件用以格式化数字，其常用的属性如表 14-6 所示。

表 14-6　<mx:NumberFormatter>组件的常用属性

属性名	说明
useThousandsSeparator	是否使用千位符 “,”。其值可为 true 或 false
useNegativeSign	是否使用负号。其值可为 true 或 false
error	格式化数据出错时的提示信息
precision	格式化数据的精度

数字格式化组件<mx:NumberFormatter>与货币格式化组件的使用方法非常相似，可以使用 useThousandsSeparator 属性来设置是否使用千位符，使用 useNegativeSign 属性来设置是否使用负号等。该组件的具体实例如代码 14.45 所示。

代码 14.45　使用<mx:NumberFormatter>组件

```
<mx:NumberFormatter id="numberFormatter" precision="4" useThousandsSepa-
rator="true" useNegativeSign="true"/>
```

14.5.5　电话格式化组件<mx:PhoneFormatter>

<mx:PhoneFormatter>组件用以格式化电话，其常用的属性如表 14-7 所示。

表 14-7　<mx:PhoneFormatter>组件的常用属性

属性名	说明
error	格式化数据出错时的提示信息
formatString	格式化掩码。例如，(###)###-####
areaCodeFormat	区号掩码。例如，(###)
validPatternChars	可用的掩码符

下面创建一个使用电话格式化组件<mx:PhoneFormatter>的实例，具体如代码 14.46 所示。

代码 14.46　使用<mx:PhoneFormatter>组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
            import mx.events.ValidationResultEvent;
            private var vResult:ValidationResultEvent;
            private function Format():void{
                vResult = pnVal.validate();
                if (vResult.type==ValidationResultEvent.VALID){
                    formattedPhone.text= phoneFormatter.format(phone.text);
                }
                else {
                    formattedPhone.text= "";
                }
```

```
            }
        ]]>
    </mx:Script>
    <mx:PhoneFormatter id="phoneFormatter" formatString="(####) ####-####"
    validPatternChars="#-()"/>
    <mx:PhoneNumberValidator id="pnVal" source="{phone}" property="text"
    allowedFormatChars=""/>
    <mx:Panel title="电话格式化组件示例" width="75%" height="75%"
          paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom=
          "10" fontSize="12">
        <mx:Form>
            <mx:FormItem label="请输入电话号码: ">
                <mx:TextInput id="phone" text="" width="190" fontSize="10"/>
            </mx:FormItem>
            <mx:FormItem label="格式化后电话号码 ">
                <mx:TextInput id="formattedPhone" text="" width="190" editable=
                "false" fontSize="10" color="#F70D44"/>
            </mx:FormItem>
            <mx:FormItem>
                <mx:Button label="验证并格式化" click="Format();"/>
            </mx:FormItem>
        </mx:Form>
    </mx:Panel>
</mx:Application>
```

在代码 14.46 中，创建了一个电话格式化组件，并设置其 formatString 属性为“(####) ####-####”，设置其 alidPatternChars 属性为“#-()”。当用户单击【验证并格式化】按钮后，首先验证用户输入的电话号码，如果输入正确就调用 PhoneFormatter 对象的 format()方法格式化数据，并在文本框中显示。具体效果如图 14-33 所示。

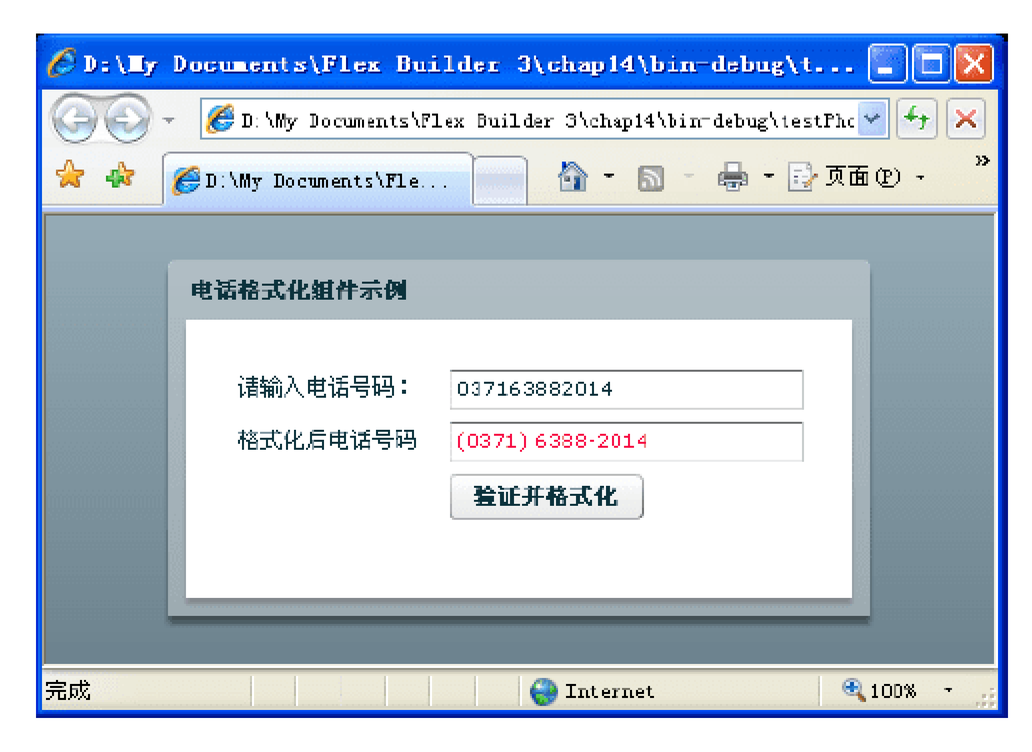


图 14-33　使用电话格式化组件

14.5.6　邮编格式化组件<mx:ZipCodeFormatter>

<mx:ZipCodeFormatter>组件用以格式化邮编，其常用的属性如表 14-8 所示。

表 14-8　<mx:ZipCodeFormatter>组件的常用属性

属性名	说明
error	格式化数据出错时的提示信息
formatString	格式化掩码。例如，####，###-###

邮编格式化组件与电话格式化组件的使用方法非常相似，需要设置其 formatString 属性。下面创建一个使用邮编格式化组件<mx:ZipCodeFormatter>的简单实例，具体如代码 14.47 所示。

代码 14.47　使用<mx:ZipCodeFormatter>组件

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
         import mx.events.ValidationResultEvent;
         private var vResult:ValidationResultEvent;
         private function Format():void{
            vResult = zcVal.validate();
            if (vResult.type==ValidationResultEvent.VALID) {
               formattedZipcode.text= zipFormatter.format(zip.text);
            }
            else {
               formattedZipcode.text= "";
            }
         }
      ]]>
   </mx:Script>
   <mx:ZipCodeFormatter id="zipFormatter" formatString="#####-####"/>
   <mx:ZipCodeValidator id="zcVal" source="{zip}" property="text" allowed-
   FormatChars=""/>
   <mx:Panel title="ZipCodeFormatter Example" width="75%" height="75%"
   fontSize="12" layout="vertical" horizontalAlign="center" verticalAlign=
   "top">
      <mx:Form width="100%">
         <mx:FormItem label="请输入 5 或者 9 位邮政编码（北美标准）:"width="100%">
            <mx:TextInput id="zip" text=""/>
         </mx:FormItem>
         <mx:FormItem label="格式化后邮政编码: " width="100%">
            <mx:TextInput id="formattedZipcode" text="" editable="false"
            color="#FD015A"/>
         </mx:FormItem>
      </mx:Form>
      <mx:Button label="验证并格式化" click="Format();"/>
   </mx:Panel>
```

```
    </mx:Application>
```

在代码 14.47 中，创建了一个邮编格式化组件，并设置其 formatString 属性为“#####-####”。当用户单击【验证并格式化】按钮后，判断用户输入的邮政编码，如果格式正确则调用 ZipCodeFormatter 对象的 format()方法进行格式化，返回格式化后的字符串并在文本框中显示，具体的效果如图 14-34 所示。

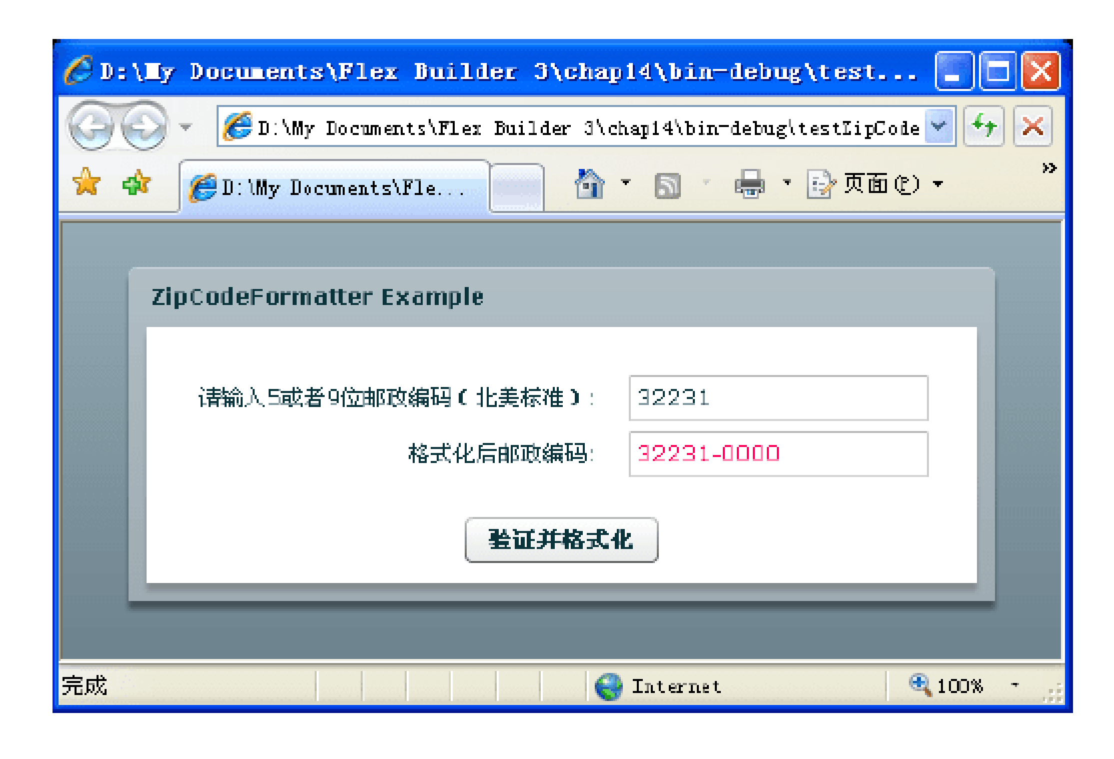


图 14-34　使用邮编格式化组件

第15章 数据传输与服务器交互



内容摘要 Abstract

数据具有流动性，数据传输是指根据用户控制传递至指定目的地。数据交互是指 Flex 与其他服务器端程序进行数据交换，包括传递数据给其他程序和接收其他程序返回的数据。本章将详细讲解数据传输的各种方法及如何与服务器进行交互。



学习目标 Objective

- 了解数据传输的几种方式
- 掌握如何使用文件流方式传输数据
- 熟悉如何使用 XML 方式传输数据
- 了解与服务器端通信的知识
- 熟悉使用 HTTPService
- 熟悉使用 WebService

15.1 数据传输的方式

Flex 3.0 中的数据传输方式包括内部数据传输、文件流方式传输、XML 方式传输、其他方式传输。应用程序的内部数据传输大多通过变量传递来实现。外部文件的数据可分为简单文本数据、XML 数据和复杂数据。对于简单的文本数据可采用文件流方式传输。对于 XML 数据可采用 XML 方式传输。对于复杂的数据，如大型数据库中的数据，需要通过其他服务器端程序来辅助传输。

15.1.1 内部数据传输

内部数据传输是指应用程序内部的数据流动，而变量传递是其最常使用的传输方式。对于同一文件或者类中的变量可采用直接赋值的方式。对于不同文件或者类中的变量可采用公有变量的方式。

1. 直接赋值方式

直接赋值是指将一变量赋值给另一变量。以下代码将变量 j 的值直接赋值给变量 i。

```
var i:int,j:int=5;
i=j;
```

大多数情况下，使用赋值运算符“=”将两变量连接起来就实现了变量传递，但在两种情况下需要特殊处理。一种情况是两个变量的类型不相同，需要强制转换。需要说明的是，若两个变量类型相近，编译器可自动转换。例如，将 int 类型的变量赋值给 Number 类型变量时，编译器自动执行变量传递。若两个变量类型相差甚远，如 Object 型与 Array 型，就需要强制转换。

Flex 3.0 中数据类型强制转换的语法如下所示。

```
变量名 as 强制类型
```

或者如下所示。

```
(强制类型)变量名
```

以下代码将 int 类型强制转换为 Number 类型。

```
var s:Number=y as Number;
var t:Number=(Number)y;
```

另一种情况是特殊的变量类型，如 Array 等多维数据变量。前面章节中详细介绍过数组变量。为了节约变量空间，Array 类型的变量并不存储全部数据，而是存储数组的首地址。若两个数组变量直接赋值，结果是两个变量都存储了同一数组的首地址，改变任何变量中的数据也就改变了数组的内容。

以下代码中两个数组变量直接赋值，带来了错误的结果。

```
var a:Array,b:Array=[4,5,6];
a=b;
a[0]=10;
trace(a);          结果: 10,5,6
trace(b);          结果: 10,5,6
```

为了帮助读者理解，假设数组在内存的首地址为 000001。变量赋值后，变量 a、b 都指向首地址 000001。对变量 a 进行数据修改后，数组数据发生改变，但变量 a、b 仍然指向同一首地址。

正确的做法是使用 concat()方法复制数组变量 b。上述代码修改后如下所示。

```
var b:Array=[4,5,6];
var a:Array=b.concat();
a[0]=10;
trace(a);          结果: 10,5,6
trace(b);          结果: 4,5,6
```

2．公有变量方式

声明变量为公有的关键字为 public。其语法如下所示。

```
public var 变量名:变量类型;
```

以下代码定义了公有变量 str。

```
public var str:String="flex";
```

不同文件或者类中使用公有变量方式传输变量，其步骤如下所示。

在项目中新建名为 Model 的文件夹，并在此文件夹下新建名为 model 的类。类中定义一公有变量 name。以下代码定义了 model 类。

```
//model.cs
package Model
{
    public class model
    {
        public static var name:String="flex 中文教程";     //定义一个静态变量
    }
}
```

在 MXML Application 文件中就可以调用 model 类的公有变量 name，如代码 15.1 所示。

代码 15.1　内部数据传输

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
fontSize="13">
    <mx:Script>
        <![CDATA[
            import Model.model;
            var sname:String=model.name;   //将"model"类中的"name"变量值传递给
                                           //"sname"变量
        ]]>
    </mx:Script>
    <mx:Panel title="内部数据传输" horizontalAlign="center" verticalAlign=
    "middle" width="288" height="148" x="109.5" y="56">
        <mx:Label text="来自 model 类的变量: {sname}"/>
    </mx:Panel>
</mx:Application>
```

15.1.2　文件流方式传输

文件流方式传输是指数据以二进制文件流的形式流动。简单的数据可存储于文本文件中，通过 Flex AIR 项目中新增的 File、FileStream 等类可以方便地操作本地文件。

创建 File 类变量的语法如下所示。

```
var File 变量:File=new File(文件路径);
```

例如定义一个 File 类变量，并指向根目录下的 test.txt 文件，具体代码如下所示。

```
var testfile:File=new File(File.applicationDirectory.nativePath+"/
test.txt");
```

其中 File.applicationDirectory.nativePath 表示项目路径。

创建 Flie 类型变量，并指向具体的文件后，接下来需要创建使用 FileStream 类打开文件。FileStream 类的语法如下所示。

```
var FileStream 变量:FileStream=new FileStream();
FileStream 变量.open(File 变量,打开方式);
```

在上述代码中，首先声明 FileStream 类的实例变量，然后使用 open()方法打开文件。其中打开方式可为 FileMode.READ、FileMode.WRITE、FileMode.APPEND、FileMode.UPDATE 4 种。例如使用 FileMode.READ 方式打开上面例子中的 test.txt 文件，就可以使用如下代码。

```
var stream:FileStream = new FileStream();   //定义 FileStream 类实例，用以处理文
                                             //件流
stream.open(testfile,FileMode.READ);         //以读的方式打开文件
```

读取 FileStream 类中的数据。在使用 FileStream 类打开文件后，数据存储于 FileStream 变量中。可使用 readUTFBytes()方法读取数据。其语法如下所示。

```
FileStream 变量.readUTFBytes();
```

ReadUTFBytes()方法返回类型为 String 型。该方法有一个可选参数 bytesAvailable，表示读取全部文件流数据。

下面创建了一个从文件中读取数据的简单实例，该实例从 text.txt 文件中读取内容，并显示在文本区域内，如代码 15.2 所示。

代码 15.2　读取文件内容

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"vertical" fontSize="13" creationComplete="initApp()" horizontalAlign=
"center" initialize="flash.system.System.useCodePage=true" verticalAlign=
"top">
   <mx:Script>
```

```
        <![CDATA[
          import flash.filesystem.*;           //引用 filesystem 下的全部类
          private var testfile:File=new File(File.applicationDirectory.
          nativePath+"/test.txt");
          private var stream1:FileStream =new FileStream();//定义 FileStream 类
          实例，用以处理文件流
            private function initApp():void
            {
                stream1.open(testfile,FileMode.READ);  //以读的方式打开文件
                txtFile.text=stream1.readUTFBytes(stream1.bytesAvailable);
                                                        //读取文件中的内容
                stream1.close();                        //关闭文件流
            }
        ]]>
    </mx:Script>
    <mx:Panel title="从文件中读取数据" verticalAlign="top" horizontalAlign=
    "center" width="446" height="200">
        <mx:TextArea id="txtFile"  width="426" height="150"/>
    </mx:Panel>
</mx:WindowedApplication>
```

运行代码 15.2，具体效果如图 15-1 所示。在图 15-2 中还给出了 test.txt 文件的详细内容。

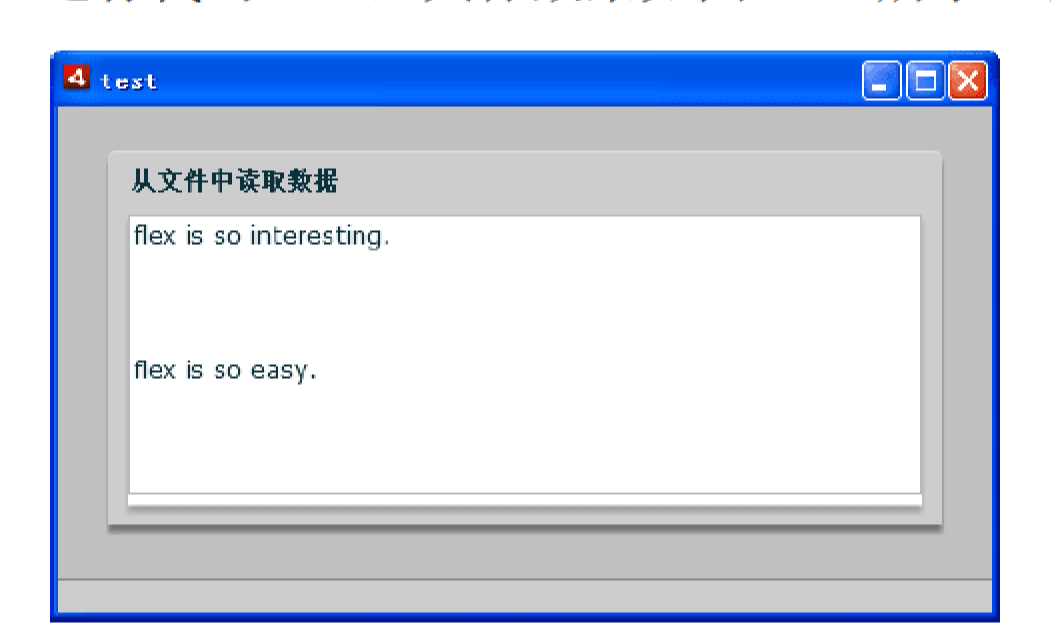


图 15-1　读取文件数据



图 15-2　test.txt 文件内容



File、FileStream 等对象只可以在 Flex AIR 项目中应用。

15.1.3　XML 方式传输

XML 是一套定义语义标记的规则，这些标记将文档分成许多部件并对这些部件加以标识。XML 允许用户自定义标记及元素，具有相当大的灵活性，不仅用户能够理解文档的内容，而且计算机也可以对其进行处理。XML 简单小巧、存储方便、检索快速的优点，使其常用于数

据存储和数据交换。在网站应用中，XML 数据应用得越来越广泛。

在 Flex 3.0 中，用户不但可以在程序内部创建 XML 类型的数据，还可以使用 URLLoader 类加载外部 XML 文件。

加载 XML 文件时需要定义一个 URLRequest 类变量，用以指明 XML 文件路径。其语法如下所示。

```
var URLRequest 变量:URLRequest=new URLRequest(XML 路径);
```

然后，就可以使用 URLLoader 类的 load 方法加载 XML 文件。其语法如下所示。

```
URLLoader 变量.load(URLRequest 变量)
```

下面创建一个简单的实例，使用 URLLoader 类加载外部文件，创建一个简单的视频播放菜单。首先来看下 tree.xml 文件的代码，如代码 15.3 所示。

代码 15.3　加载外部 XML 文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<node label="视频列表">
    <node label="第 1 章  Flex 基础" data="">
        <node label="认识 Flex 3.0" data="images/1.flv"/>
        <node label="Flash CS3 环境配置" data="images/2.flv"/>
             <node label="Flex Builder 环境搭建" data="images/3.flv"/>
    </node>
    <node label="第 2 章  熟悉开发环境 Flex Builder 3" data="">
             <node label="熟悉 Flex Builder 3 工作区" data="images/4.flv"/>
        <node label="编译与运行 Flex 3.0 程序详解" data="images/5.flv"/>
        <node label="调试 Flex 3.0 程序" data="images/6.flv"/>
        <node label="Flex 3.0 项目项目概述" data="images/7.flv"/>
        <node label="Flex Builder 3 常用快捷键" data="images/8.flv"/>
    </node>
    <node label="第 3 章  ActionScript 3.0 语法" data="">
        <node label="变量和常量" data="images/9.flv"/>
        <node label="数据类型" data="images/10.flv"/>
             <node label="运算符" data="images/11.flv"/>
             <node label="流程语句" data="images/12.flv"/>
    </node>
</node>
```

接下来设计播放菜单界面，并使用具体的功能代码获取 tree.xml 文件中的数据，如代码 15.4 所示。

代码 15.4　播放菜单界面

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
initialize="flash.system.System.useCodePage=true" creationComplete=
```

```
"OperateXML()">
    <mx:Script>
       <![CDATA[
           [Bindable]
           public var selectedNode:XML;
           [Bindable]
           public var treenew:XML;
           public function treeChanged(event:Event):void {
              selectedNode=Tree(event.target).selectedItem as XML;
              }
             public function OperateXML():void{
             var testloader:URLLoader=new URLLoader();
             var req:URLRequest=new URLRequest("tree.xml")
             testloader.dataFormat=URLLoaderDataFormat.TEXT;
             //添加事件监听
             testloader.addEventListener(Event.COMPLETE,handleComplete);
             testloader.load(req);
             }
             //XML 文档加载完成后的处理函数
             public function handleComplete(event:Event):void{
              try{
               treenew=new XML(event.target.data);
              }
              catch(e:TypeError){
               trace("Could not parse text into XML");
               trace(e.message);
              }
             }
  ]]>
   </mx:Script>
   <mx:Panel height="99%" layout="absolute" width="364" title="播放列表"
   fontSize="12" x="159" y="10">
            <mx:Tree  id="tree1" width="313" dataProvider="{treenew}"
            labelField="@label" height="100%" change="treeChanged(event)"
            showRoot="false" x="10">
            </mx:Tree>
    </mx:Panel>
</mx:Application>
```

在代码 15.4 中，在页面初始化函数 OperateXML()中，创建了一个 URLLoader 和 URLRequest 对象实例。在声明 URLRequest 对象时获取外部 XML 文件的路径，使用 URLLoader 对象的 load()方法进行加载，同时对外部 XML 文件加载完成事件的监听。一旦加载完成执行 completeHandle()函数。

在函数 completeHandle()中，设置 XML 类型变量 treenew 获取返回的 XML 数据列表。并

且在 Tree 组件中，设置其 dataProvider 属性为{treenew}，设置 labelField="@label"，即显示数据源节点的 Label 属性值，并且设置其 change 事件调用函数 treeChange(event)，获取用户选择结果。

运行代码 15.4，具体的显示效果如图 15-3 所示。

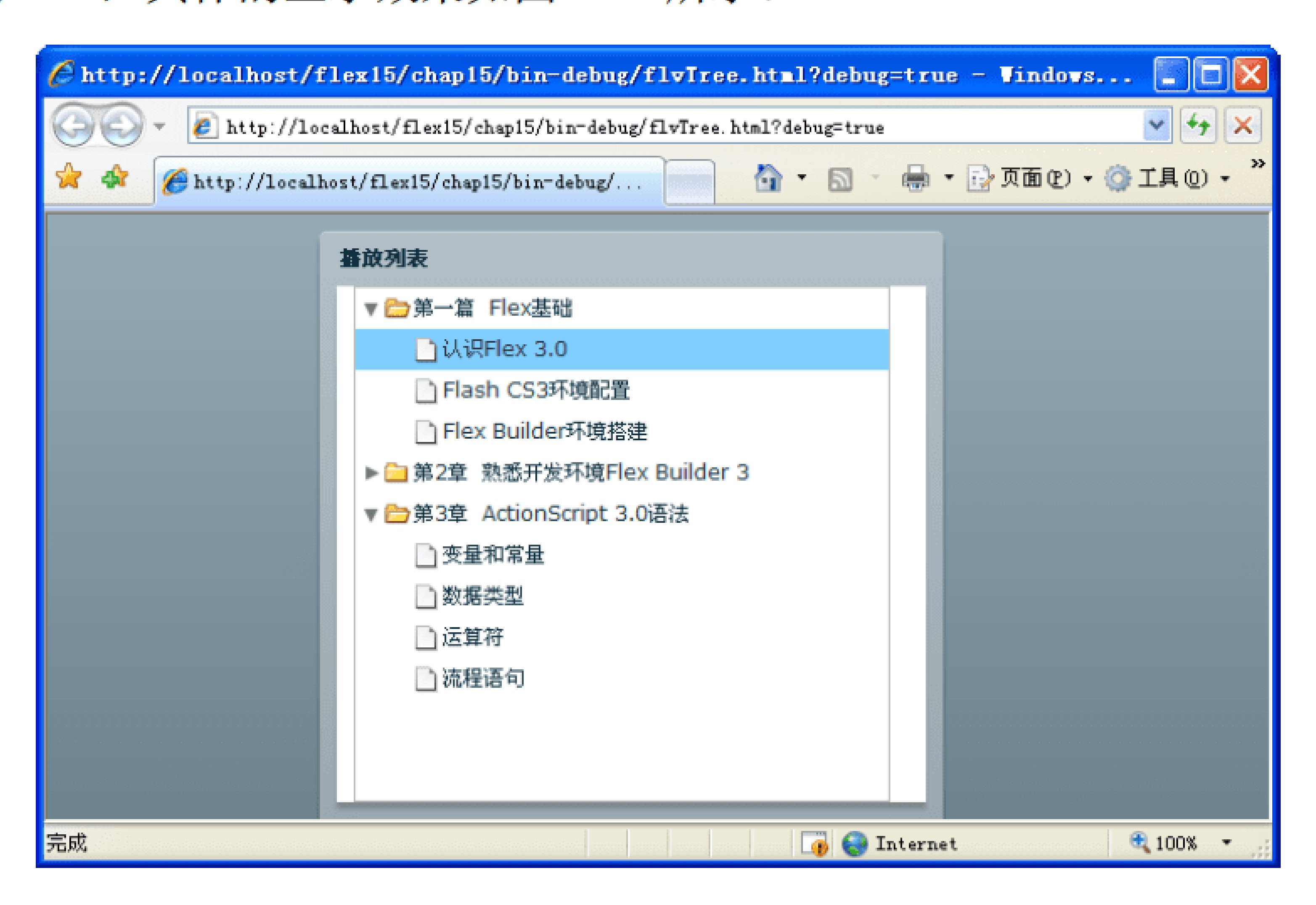


图 15-3　读取外部 XML 文件

15.1.4　其他方式传输

除了上述小节中介绍的数据外，Flex 应用程序可能还会遇到其他类型的外部数据，如 SQL Server 数据库数据、MySQL 数据库数据、Oracle 数据库数据等。Flex 3.0 不能直接接收这些数据，需要与其他服务器端程序进行交互，在服务器端处理这些数据并以特定的类型，如数组型、XML 型、Object 型传递给 Flex。

Flex 主要是用于前端用户交互的解决方案框架，但是作为完整的信息系统来说，存储数据和读取数据等数据操作是必不可少的内容。使用 Flex 开发业务逻辑复杂的信息系统时，通常会选用成熟的服务器端架构。Flex 提供了一系列与服务器交互的方式，通过这些方式可以方便地实现与成熟的服务器端架构进行通信。

Flex 定位于企业级开发平台，所以与服务器端通信是 Flex 中最重要的功能。Flex 与服务器端通信分为两大类，一类是使用传统的 HTTP 方式发送请求进行与服务器端的通信，另外一类是通过 LCDS 与服务器端交互。这两大类与服务器端通信的方式在技术上又有着不同的实现方式。本书主要介绍与 ASP.NET 服务器端开发技术进行交互，因此在本章就只介绍如何使用传统的 HTTP 方式与服务器端进行交互。

使用传统的 HTTP 方式与服务器通信可以分为两大类。

- **HTTPService**　使用异步的 HTTP 请求方式与服务器端进行通信，其使用方式与 Ajax 一样。

❑ **WebService** 使用标准的 WebService 协议与服务器端进行通信。

采用 HTTP 方式与服务器端进行通信时，开发 Flex 端程序与服务器端程序分开，也就是说，Flex 内容是独立的，只需要引入相应的标签和类库就可以访问服务器端的数据。下面将分别对这两种使用 HTTP 方式与服务器端通信的方式进行讲解。

15.2 使用 HTTPService 与服务器端交互

在 Flex 中，可以通过组件以及类库的方式访问 HTTPService，两者的使用方式几乎一致。HTTPService 不是标准协议，所以在使用 HTTPService 进行访问的时候，需要自己定义数据交换格式。

1. HTTPService 类

在 Flex 3.0 的 mx.rpc.http 下提供了一个 HTTPService 类。使用 HTTPService 对象与服务器端通信时，就会使用该类进行数据访问。HTTPService 类常用的属性及方法如表 15-1 所示。

表 15-1 HTTPService 常用的属性和方法

名称	类型	说明
contentType	属性	String 类型，默认值为 application/x-www-form-urlencoded。表明请求内容的类型
destination	属性	String 类型，表明在 LCDS 当中 service-config.xml 描述的 HTTP 服务的别名
headers	属性	Object 类型，发送请求定义的 HTTP 头
mothod	属性	Stirng 类型，默认值为 GET。表明发送 HTTP 请求的方式，取值为 POST 或者 GET
request	属性	Object 类型，发送请求的参数。内容格式为名称和值对
requestTimeout	属性	int 类型，请求超时的时间。单位为秒
rootURL	属性	String 类型，请求 HTTP 服务的根地址
url	属性	String 类型，请求 HTTP 服务的位置
userProxy	属性	Boolean 类型，是否使用代理的别名服务
xmlDecode	属性	Function 类型，当请求为 XML 时，处理结果 XML 的 ActionScript 函数
xmlEncode	属性	Function 类型，对发送请求内容进行 XML 化处理的 ActionScript 函数
HTTPService()	方法	构造方法
disconnect()	方法	没有返回值，断开 HTTP 服务的网络连接
send()	方法	参数是 Object 类型，默认为 null，返回值类型为 AsyncToken。执行一个 HTTPService 的请求

Flex 对 HTTPService 访问的方式与传统的 B/S 结构很相似，同样是通过 POST 和 GET 方式发送请求。不同之处在于，Flex 发送请求和处理结果的方式是异步的，而不是传统 B/S 结构那样同步进行处理。

下面给出了一个使用 HTTPService 类的简单实例。

```
var objHTTP:HTTPService=new HTTPService();
objHTTP.url="../dataBase/test.aspx";
objHTTP.method="GET";
objHTTP.addEventListener(ResultEvent.RESULT,Resulthandler);
objHTTP.addEventListener(FaultEvent.FAULT,faultHandler);
var oArgs:Object = {};  //定义一个参数列表，以 GET 方式提交
oArgs["EditActionID"]=indexID;
oArgs["EditType"]=dsFieldName;
oArgs["EditValue"]=newValue;
oArgs["rndNum"]=Math.random();
objHTTP.send(oArgs);
```

在上述代码中，首先创建了 HTTPService 的一个实例对象 objHTTP，设置其 url 属性指向服务器端程序文件的位置，设置以 GET 方式发送 HTTP 请求，并且创建了一个参数列表，最后调用 send()方法发送 HTTPService 请求。

HTTPService 请求返回的数据存储于 ResultEvent 类中。在上述代码中，添加了对 result 事件和 Fault 事件的监听，并指定了结果和错误处理函数。在结果处理函数中，就可以根据从服务器端返回的数据，进行具体的处理。例如：

```
private function Resulthandler(e:ResultEvent):void
{
    e.result        //返回数据
}
```

2．使用<mx:HTTPService>组件

虽然可以通过类的方式直接访问 HTTPService，但是通常在 Flex 中，用户会使用<mx:HTTPService>组件的形式对 HTTPService 进行访问。<mx:HTTPServcie>组件在 mx.rpc.http.mxml 包下。

<mx:HTTPService>组件的属性和方法与 HTTPService 类的完全相同，可参考表 15-1。下面通过实例来介绍该组件的具体用法。

```
<mx:HTTPService id="addmessage" url="../dataBase/test.aspx" resultFormat=
"e4x" method="GET" result="resultHandler(event)" fault="faultHandler
(event)">
      <mx:request>
          <theme>{txt_theme.text}</theme>
          <content>{txt_content.text}</content>
          <username>{guestname.text}</username>
      </mx:request>
</mx:HTTPService>
```

在上述代码中，设置 HTTPService 的 id、url、method 等属性，并且指定了错误处理函数和结果处理函数。在<mx:request>组件中定义了 3 个参数，这样用户在使用时，就可以直接调

用 HTTPService 对象的 send()方法，而无须在 send()方法中重新设置参数。

HTTPService 对象还可以发送 XML 数据，发送 XML 数据有两种实现方法，分别如下所示。

❑ **把 XML 数据作为 request 对象的一个属性**

例如：

```
var paras:Object=new Object();
paras.info="<info><name>admin</name><pwd>123456</pwd></info>"
src.send(paras);
```

在上述代码中创建了一个 Object 对象 paras，并且设置其 info 属性为一段 XML 类型数据，最后调用 HTTPService 对象的 send()方法发送该对象 paras。

❑ **以 XML 形式发送数据**

例如：

```
var strxml:XML=<info>
             <username>admin</username>
             <pwd>123456</pwd>
             </info>
 src.send(strxml);
```

以 XML 形式发送数据时，必须设定 HTTPService 对象的 contentType 属性为 application/xml，默认情况下是 application/x-www-form-urlencoded，例如如下代码：

```
<mx:HTTPService id="src" url="login.aspx" contentType="application/xml"
method="POST" result="resulthandler(event)" fault="faulthandler(event)" />
```

使用 XML 数据，便于处理结构复杂的数据，而且 XML 格式符合 Web 标准，所有的服务器端语言都能够方便地解析，使得程序有良好的移植性。

发送 HTTP 请求后，就可以对服务器端返回的数据进行处理。结果处理函数和错误处理函数与使用 HTTPService 类库时完全相同，这里就不再重复。

15.3 HTTPService 应用实例——留言本

在日常的网站应用中，留言本模块是一个非常常见的模块。留言本是网站管理人员和用户、用户与用户之间交流的常用平台。本小节根据前面掌握的知识，使用 HTTPServervice 方式与数据库交互，创建一个简单的留言本应用实例。

15.3.1 编写 ASP.NET 程序

Flex 对当前主流的服务器端开发技术 ASP.NET、PHP、J2EE 都进行了集成开发支持，用

户可以根据需要选择合适的服务器端开发技术。在本实例中，将应用 ASP.NET 服务器端开发技术。

1. 数据库设计

在创建留言本之前，首先需要对该留言本所涉及到的功能进行需求分析，接下来需要创建数据库，存放该实例中所涉及到的数据。

在本实例中，使用 Access 数据库来存放数据。在数据库中，设计了两个数据表：存放留言信息的 MessageInfo 表和存放管理员账户信息的 admininfo 表。这两个表的字段、类型及说明分别如表 15-2、表 15-3 所示。

表 15-2　MessageInfo 表

字段名	字段类型	说明
id	自动编号	唯一标识每一条留言信息
userName	文本	留言人姓名
theme	文本	留言信息的主题
content	备注	留言信息的主内容

表 15-3　admininfo 表

字段名	字段类型	说明
id	自动编号	唯一标识每一条账户信息
adminname	文本	管理员账户
pwd	文本	管理员密码

2. 验证管理员登录

数据表创建完成后，就可以在服务器端创建 ASP.NET 程序代码。首先创建用于验证管理员账户正确性的代码，如代码 15.5 所示。

代码 15.5　验证管理员信息

```
protected void Page_Load(object sender, EventArgs e)
    {
        string constr ="Provider=Microsoft.Jet.OLEDB.4.0; Data Source="+Server.
        MapPath("data.mdb") + ";";
        OleDbConnection conn = new OleDbConnection(constr);
        string name = Request.QueryString["name"].ToString();
        string pwd = Request.QueryString["pwd"].ToString();
        string sql = "select * from admininfo where adminname='" + name + "'
        and pwd='" + pwd + "'";
        OleDbCommand com = new OleDbCommand(sql, conn);
        com.Connection.Open();
        OleDbDataReader dr = com.ExecuteReader();
```

```
        if (dr.Read())
        {
            Response.Write("<result><state>ok</state></result>");
        }
        else
        {
            Response.Write("<result><state>false</state></result>");
        }
        dr.Close();
        conn.Close();
    }
```

在代码 15.5 中，使用 Request.QueryString()方法获取 HTTPService 请求中的管理员账户和密码等信息。然后连接数据库，定义具体的 SQL 语句进行查询，判断当前获取到的账户和密码信息在数据表中是否存在，并返回相应的类似 XML 类型的字符串。在 Flex 程序中，就可以根据相应的返回结果，进行具体的处理。

3．显示、添加和删除留言

留言信息是留言本的核心，因此对留言信息的操作就显得尤其重要。在本实例中，定义了显示当前所有留言、添加留言、删除留言等操作的具体代码，如代码 15.6 所示。

代码 15.6　操作数据

```
<script language="C#" runat="Server">
    OleDbDataReader dr;
    OleDbConnection conn;
    OleDbDataAdapter da;
    DataSet ds;
    string ConnStr, xmlStr;
    public void Page_Load(Object src, EventArgs e)
    {
        if (Request.QueryString["DelActionID"] != null)
        {
            string itemID = Request.QueryString["DelActionID"].ToString();
            DeleteItem(itemID);
        }
       if (Request.QueryString["Action"] != null){
            string userName = Request.QueryString["username"];
                                  //这里就是获取 Flex 提交过来的 username 变量
            string theme = Request.QueryString["theme"];
                                  //同理，这里是获取 Flex 提交过来的 theme 变量
            string content = Request.QueryString["content"];
            InsertItem(userName, theme,content);
        }
```

```
            ShowAllUserInfo();
        }
        //显示所有留言信息
        private void ShowAllUserInfo()
        {
            ConnStr = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" + Server.
            MapPath("data.mdb") + ";";
            string strSQL = "select * from MessageInfo";
            OleDbConnection MyConn = new OleDbConnection(ConnStr);
            OleDbCommand MyComm = new OleDbCommand(strSQL, MyConn);
            MyComm.Connection.Open();
            dr = MyComm.ExecuteReader();
            xmlStr = "<Message>";
            while (dr.Read())
            {
                xmlStr += "<MessageInfo><id>" + dr["id"].ToString() + "</id>";
                xmlStr += "<userName>" + dr["userName"].ToString() + "</userName>";
                xmlStr += "<Theme>" + dr["Theme"].ToString() + "</Theme>";
                xmlStr += "<content>" + dr["content"].ToString() + "</content>";
                xmlStr += "</MessageInfo>";
            }
            xmlStr += "</Message>";
            dr.Close();
            MyConn.Close();
            Response.Write(xmlStr);
        }
        //向数据库中插入留言信息
        private void InsertItem(string newName, string theme,string content)
        {
            ConnStr = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" + Server.
            MapPath("data.mdb") + ";";
            string strSel = "INSERT INTO MessageInfo(userName,Theme,content)
            VALUES('" + newName + "','" + theme + "','" + content + "')";
            OleDbConnection MyConn = new OleDbConnection(ConnStr);
            OleDbCommand MyComm = new OleDbCommand(strSel, MyConn);
            MyComm.Connection.Open();
            dr = MyComm.ExecuteReader();
            dr.Close();
            MyConn.Close();
        }
    //删除留言信息
        private void DeleteItem(string itemID)
        {
            ConnStr = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" + Server.
            MapPath("data.mdb") + ";";
```

```
            string strSel = "delete from MessageInfo where id=" + itemID;
            OleDbConnection MyConn = new OleDbConnection(ConnStr);
            OleDbCommand MyComm = new OleDbCommand(strSel, MyConn);
            MyComm.Connection.Open();
            dr = MyComm.ExecuteReader();
            dr.Close();
            MyConn.Close();
        }
    </script>
```

在代码 15.6 中，首先定义了一些经常使用的变量对象，然后创建了 3 个函数：ShowAllUserInfo()、InsertItem()、DeleteItem()。最后程序初始化时，根据获取到的 HTTP 请求中的参数值，判断、执行相对应的函数，完成特定的数据操作。

- 在函数 ShowAllUserInfo()中，使用 OledbDataReader 对象的 Read()方法获取 MessageInfo 表的所有信息，存放在变量 XmlStr 中，并且使用 Response.Write()方法返回。
- 在函数 InsertItem()中，定义了 3 个参数：newName、theme、content，分别代表留言人的姓名、主题、内容。根据获取 HTTPService 请求中的 userName、theme、content 参数值，执行 SQL 语句，向数据库中插入数据。
- 在函数 DeleteItem()中，定义一个参数 itemID，根据获取 HTTPservice 请求中的 DelActionID 参数值，执行 SQL 语句，删除数据库中的相应记录。

15.3.2 创建虚拟目录

服务器端代码设计完成后，下面来创建基于 ASP.NET 服务器端的 Flex 项目程序。基于 ASP.NET 服务器端的 Flex 应用运行在 Microsoft 公司的 IIS（Internet Information Service）上。IIS 是由 Microsoft 公司随着 Windows 操作系统发布的一款 Web 服务器。同时，它也是运行 ASP 和 ASP.NET 程序的应用服务器。如果用户的机器上没有安装 IIS，可以通过添加 Windows 组件的方式安装 IIS 服务器。

在创建 Flex 项目之前，首先为该 Flex 项目的根文件夹设置虚拟目录，具体过程如下所示。

（1）首先，在【Internet 信息服务】对话框中右击【默认网站】，从弹出的快捷菜单中选择【新建】|【虚拟目录】命令，打开【虚拟目录新建向导】对话框。

（2）单击【下一步】按钮，打开【创建虚拟目录名】对话框，如图 15-4 所示。

在【创建虚拟目录名】对话框中仅有一个文本框，该文件框的作用是指定虚拟目录的名称，这在以后访问该目录下的文件时需要用到。

（3）在输入完虚拟目录的名称后，单击【下一步】按钮，打开如图 15-5 所示的【网站内容目录】对话框。在【网站内容目录】对话框中，需要为新建的虚拟目录指定相对应的本地路径，这样就将本地路径与虚拟目录相联系起来。

（4）单击【下一步】按钮，打开如图 15-6 所示的【访问权限】对话框。为了保证网站的安全，通常只需要保留默认的选项即可。

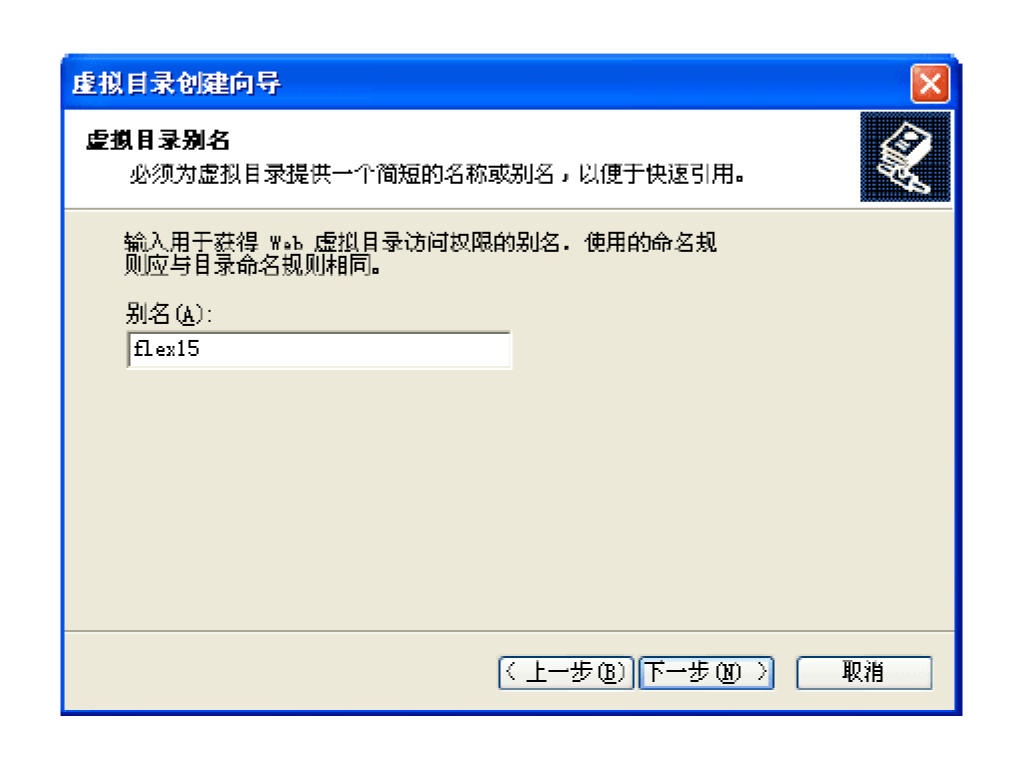


图 15-4 【创建虚拟目录名】对话框

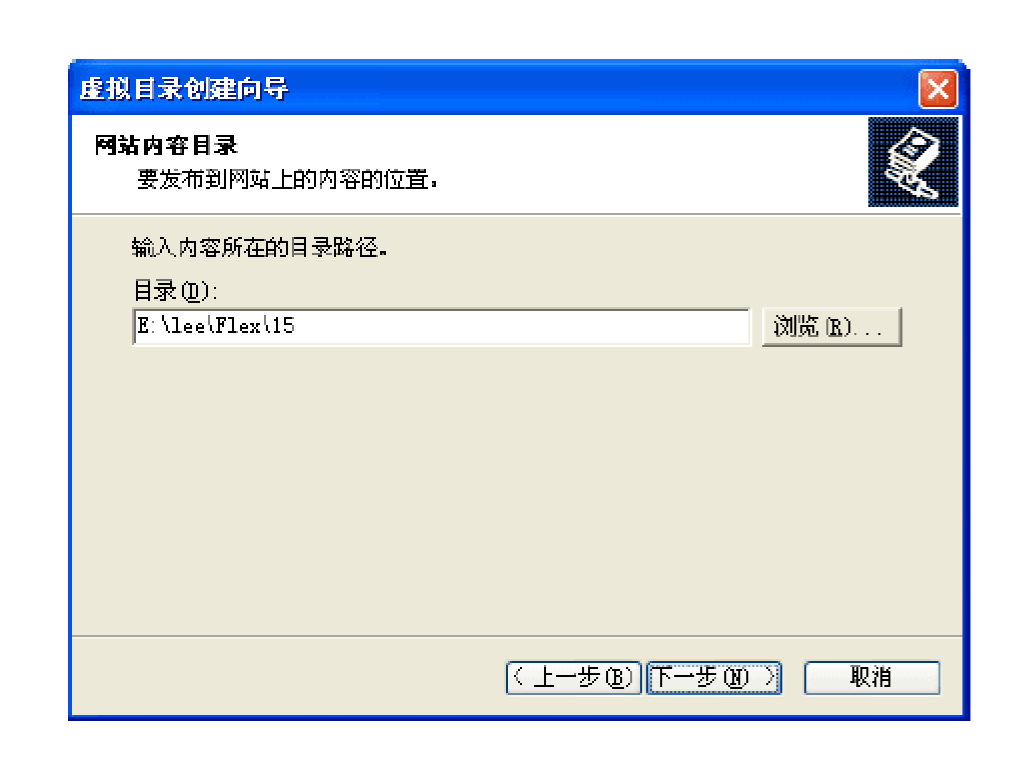


图 15-5 【网站内容目录】对话框

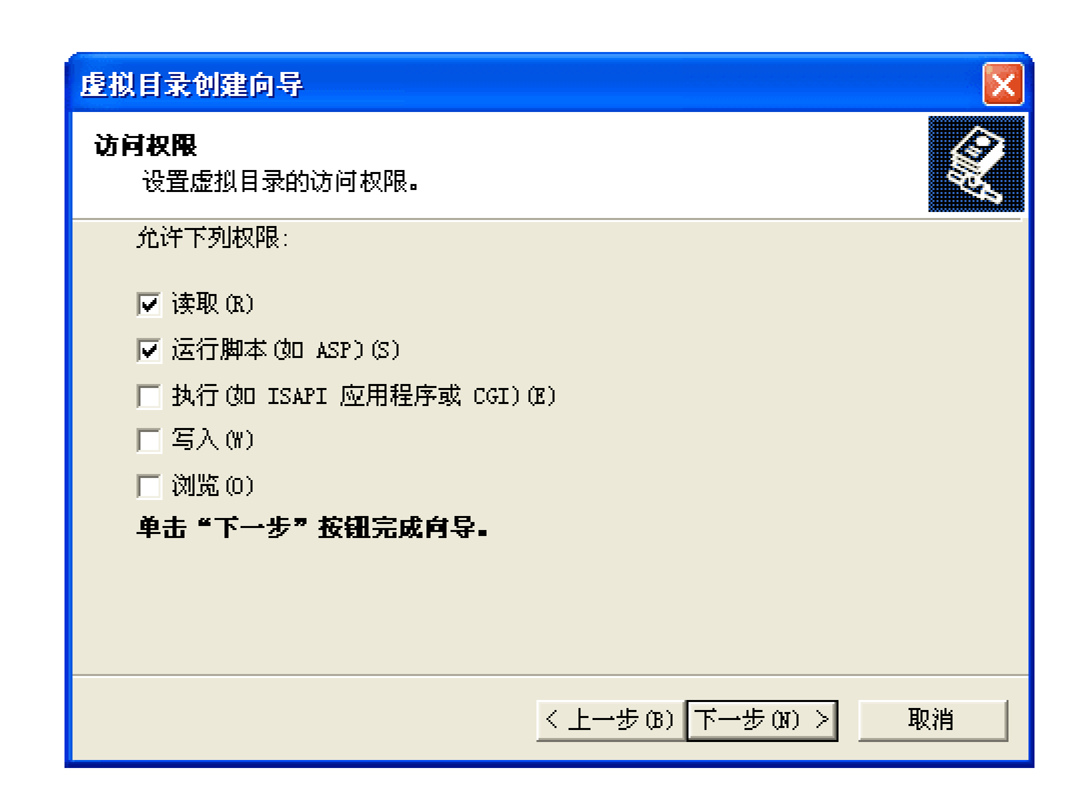


图 15-6　访问权限设置

（5）接着单击【下一步】按钮，完成虚拟目录的创建。虚拟目录创建完成后，就可以在该文件夹下创建基于 ASP.NET 服务器端技术的 Flex 项目。

15.3.3　留言本界面与功能实现

在 15.3.2 小节中已经配置了 IIS 服务器环境，创建了虚拟目录，那么接下来就进行具体的界面设计。在该实例中，将留言本功能分为 4 个简单的模块：留言信息列表模块、添加留言模块、查看留言详细信息模块、管理员登录并管理留言模块。本节将分别对各个模块的布局及功能实现进行详细地讲解。

1．界面框架布局

在网站的开发过程中，页面的布局非常重要，一个优秀的页面布局是网站成功的一半。在本实例中，使用 VDividedBox 组件将页面垂直分割成为两部分：第一部分在页面的顶部显示网站的 logo 信息，并创建一些快捷命令，比如管理员登录等；第二部分是留言本的主内容模块，在该模块中，使用 ViewStack 组件创建不同的视图，每个视图代表具体的功能模块。页面的布局代码如代码 15.7 所示。

代码 15.7　留言本界面框架

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout=
"horizontal"  horizontalAlign="center" verticalAlign="top" creationComplete=
"initApp()">
    <mx:Script source="xxLib.as" />
      <mx:states>
       <mx:State name="admin">
          <mx:AddChild relativeTo="{list}" position="lastChild">
             <mx:target>
                <mx:Button id="but_del" label="删除留言" click="deleteItem()"
                x="243" y="320"/>
             </mx:target>
          </mx:AddChild>
          <mx:SetProperty target="{link1}" name="visible" value="fasle"/>
          <mx:SetProperty target="{link2}" name="visible" value="fasle"/>
          <mx:SetProperty target="{but_add}" name="visible" value="fasle"/>
        </mx:State>
    </mx:states>
    <mx:VDividedBox width="800">
        <mx:VBox width="100%" height="120" backgroundImage="banner.gif"
        verticalAlign="bottom" >
            <mx:HBox width="800" horizontalAlign="right" color="#E26539">
               <mx:Label id="username" text=""/>
                <mx:LinkButton id="link1" label="写新留言" click="v1.selected
                Child=add;" />
                <mx:LinkButton id="link2" label="管理员登录" click="showLogin();"/>
            </mx:HBox>
        </mx:VBox>
        <mx:VBox width="100%" height="650">
            <mx:ViewStack id="v1" width="100%" height="100%">
            //…这里是具体的子级容器内容
            </mx:ViewStack>
        </mx:VBox>
    </mx:VDividedBox>
</mx:Application>
```

在代码 15.7 中，使用<mx:Script source="xxLib.as" />，引用了外部.as 文件，该文件定义了各个功能模块所使用到的所有功能代码。在 VDividedBox 组件上，定了一个新的状态 admin，当管理员登录成功时显示该状态界面。在该状态中，创建了一个新的按钮，用于删除留言信息，设置【写新留言】、【管理员登录】和【新增】按钮不可见，并且修改 Label 组件的 text 属性，显示管理员账号信息。

在顶部 logo 信息部分，创建了两个 LinkButton 组件，创建了【写新留言】和【管理登录】

两个快捷命令，并分别设置其 click 事件。

在 xxLib.as 文件中添加代码，具体内容如代码 15.8 所示。

代码 15.8　添加程序应用和创建公用函数

```
//添加引用
import mx.controls.Alert;
import mx.events.CloseEvent;
import mx.events.DataGridEvent;
import mx.rpc.events.FaultEvent;
import mx.rpc.events.ResultEvent;
import mx.rpc.http.mxml.HTTPService;
[Bindable]
public var indexname:String;
[Bindable]
public var indextheme:String;
[Bindable]
public var indexcontent:String;
public function exitto():void{
    v1.selectedChild=listall;
}
public function showLogin():void{
    v1.selectedChild=adminlogin;
}
```

在代码 15.8 中，添加了一些引用、声明了 3 个全局变量和创建了两个函数。其中，函数 exitto()设置 ViewStack 组件的 selectedChild 属性为 listall，即返回到留言列表界面；函数 showLogin()设置 ViewStack 组件的 selectedChild 属性为 adminlogin，即进入管理员登录界面。

2. 留言列表界面

留言列表界面是用户进入留言的主界面。在该界面中，使用 DataGrid 组件显示所有留言信息列表。在 DataGrid 组件中创建了 4 列：编号、姓名、主题和内容，并且分别设置每一列的 dataField 属性。在 DataGrid 组件下面创建了两个 Buttoon 按钮，分别用于新增留言和查看详细留言信息。具体代码如代码 15.9 所示。

代码 15.9　留言信息列表界面

```
<mx:Panel id="listall" width="800"  title="我的留言列表" layout="absolute">
    <mx:Canvas id="list" horizontalCenter="0" top="0">
     <mx:DataGrid id="dgData" width="600" height="300" editable="true">
       <mx:columns>
        <mx:DataGridColumn headerText="编号" dataField="id" width="50"
        editable="false" />
        <mx:DataGridColumn headerText="姓名" dataField="userName" width="100"
```

```
        editable="false" />
        <mx:DataGridColumn headerText="主题" dataField="Theme" width="150"
        editable="false"  />
        <mx:DataGridColumn headerText="内容" dataField="content"  editable=
        "false" itemEditor="mx.controls.TextArea"/>
    </mx:columns>
   </mx:DataGrid>
    <mx:Button id="but_add" label="新增" x="243" y="320" click="v1.selected-
    Child=add;"/>
    <mx:Button label="查看详细信息" x="62" y="320" click="selectall()"/>
    </mx:Canvas>
</mx:Panel>
```

在创建 DataGrid 组件时，并没有指定相应的数据源。那么在页面加载时，就需要向服务器端发送请求，获取数据源，如代码 15.10 所示。

代码 15.10　获取留言信息

```
//声明 HTTPService 对象实例
private  var objHTPS:HTTPService=new HTTPService();
//页面初始化函数
private function initApp():void{
    objHTPS.url="../dataBase/save_mdb.aspx";
    objHTPS.resultFormat="e4x";
    objHTPS.addEventListener(ResultEvent.RESULT,ShowMessageList);
    objHTPS.addEventListener(FaultEvent.FAULT,faultHandler);
    objHTPS.send();
}
//结果处理函数
private function ShowMessageList(e:ResultEvent):void{
    dgData.dataProvider=e.result.MessageInfo;
}
//错误处理函数
private function faultHandler(e:FaultEvent):void{
    var err:String="发生错误";
    mx.controls.Alert.show(e.message.toString(),err);
}
```

在代码 15.10 中，创建了函数 initApp()，当页面加载时调用。在函数 initApp()中，创建了 HTTPService 对象的实例 objHTPS，设置该实例对象的 url 属性，并指定回调函数为 ShowMessageList()，错误处理函数为 faultHandler()。在函数 ShowMessageList()中，设置 DataGrid 组件的 dataProvider 属性，配置数据源。在函数 faultHandler()中，弹出具体的错误信息。

留言列表界面的具体显示效果如图 15-7 所示。



图 15-7　留言列表

3. 新增留言界面

当用户单击【写新留言】命令或者【新增】按钮时，就会进行【新增留言】界面。在该界面中，使用 Form 表单组件进行布局，接受用户输入的留言信息。具体布局代码如代码 15.11 所示。

代码 15.11　新增留言界面

```
<mx:Canvas id="add">
    <mx:Panel width="800" layout="horizontal" title="写留言" fontSize="12"
    color="#263EF3" horizontalAlign="center" verticalAlign="top">
        <mx:Form width="600" height="430" borderStyle="solid">
            <mx:FormHeading label="写新留言" width="100%" color="#000000"/>
            <mx:FormItem label="主题: "  color="#000000" width="543">
                <mx:TextInput id="txt_theme" width="441"/>
            </mx:FormItem>
            <mx:FormItem label="内容: "  color="#000000" width="543">
                <mx:RichTextEditor id="txt_content" title="" width="441"
                height="200">
                </mx:RichTextEditor>
            </mx:FormItem>
            <mx:FormItem label="你的姓名: "  color="#000000" width="543">
                <mx:TextInput id="guestname" width="441" text="游客"/>
                <mx:CheckBox id="chk1" selected="true" label="匿名留言"click=
                "show()" />
```

```
            </mx:FormItem>
            <mx:FormItem  color="#000000" width="542">
              <mx:HBox>
                <mx:Button id="sumbit" label="提交留言" click="addmessage.
                send()" />
                <mx:Spacer width="100" />
                <mx:Button id="cancle" label="取消" click="exitto()" />
              </mx:HBox>
            </mx:FormItem>
        </mx:Form>
    </mx:Panel>
</mx:Canvas>
```

在代码 15.11 中，创建了两个命令按钮，【提交留言】和【取消】按钮。其中，【取消】按钮调用函数 exitto()，返回留言信息列表；按钮【提交留言】，调用 HTTPService 对象的 Send()方法，将用户输入的留言信息发送到服务器端进行处理。这里首先创建了一个 HTTPService 对象，如代码 15.12 所示。

代码 15.12　创建新增留言的 HTTPService 对象

```
<mx:HTTPService id="addmessage" url="../dataBase/save_mdb.aspx" resultFormat=
"e4x" method="GET" result="myHandler(event)" showBusyCursor="true" fault=
"faultHandler(event)">
   <mx:request>
       <theme>{txt theme.text}</theme>
      <content>{txt content.text}</content>
      <username>{guestname.text}</username>
      <Action>add</Action>
   </mx:request>
</mx:HTTPService>
```

在代码 15.12 中创建了一个 HTTPService 对象，将用户输入的留言信息作为参数发送给服务器端。声明 url 属性，并且指定结果处理函数为 myHandler()，错误处理函数为 faultHandler()。

下面列出了该界面的具体功能代码，如代码 15.13 所示。

代码 15.13　实现新增留言功能

```
//判断是否匿名登录
private function show():void{
      if(chk1.selected){
      guestname.text="游客";
      guestname.editable=false;
      }
      else{
      guestname.text="";
```

```
            guestname.editable=true;
            }
    }
    //新增留言的结果处理函数
    private function myHandler(e:ResultEvent):void{
        var returnValue:String=e.result.toString();
        dgData.dataProvider=e.result.MessageInfo;
        objHTPS.disconnect();
        Alert.show("添加成功,返回列表? ","提示信息",Alert.YES|Alert.NO,this,
        selecthandler,null,1);
        }
    //警告对话框结果处理函数
    private function selecthandler(event:CloseEvent):void{
      if(event.detail==Alert.YES){
      exitto();
      }
      else{
       txt_theme.text="";
       txt_content.text="";
       guestname.text="";
      }
    }
    //添加留言失败时的错误处理函数
    private function faultHandler(e:FaultEvent):void{
        var err:String="发生错误";
        mx.controls.Alert.show(e.message.toString(),err);
    }
```

在代码 15.13 中，创建了 4 个函数。其中，函数 show()，用于控制留言人姓名不可以为空，当用户启用【匿名留言】选项时，默认留言人姓名为“游客”；函数 myHandler()，获取服务器返回结果，添加成功后显示提示界面跳转信息；函数 selecthandler()用于监听用户的选择，如果用户选择 Yes，则调用函数 exitto()，返回到留言信息列表界面，如果用户选择 NO，则清空文本框中所有信息，继续添加留言；函数 faultHandler()用于当发生错误时，显示相应的错误信息。

新增留言界面的具体效果如图 15-8 所示。

当用户选择 Yes 按钮后，返回到留言信息列表，即可看到新增的留言信息已经添加成功，如图 15-9 所示。

4．查看留言详细信息界面

在留言列表界面，列出了所有的留言信息。但是由于页面大小有限，不可以把所有的留言内容都显示出来。这时就可以选择需要查看的留言，单击【查看详细信息】按钮，打开查看留言详细信息界面。该界面的布局代码如代码 15.14 所示。

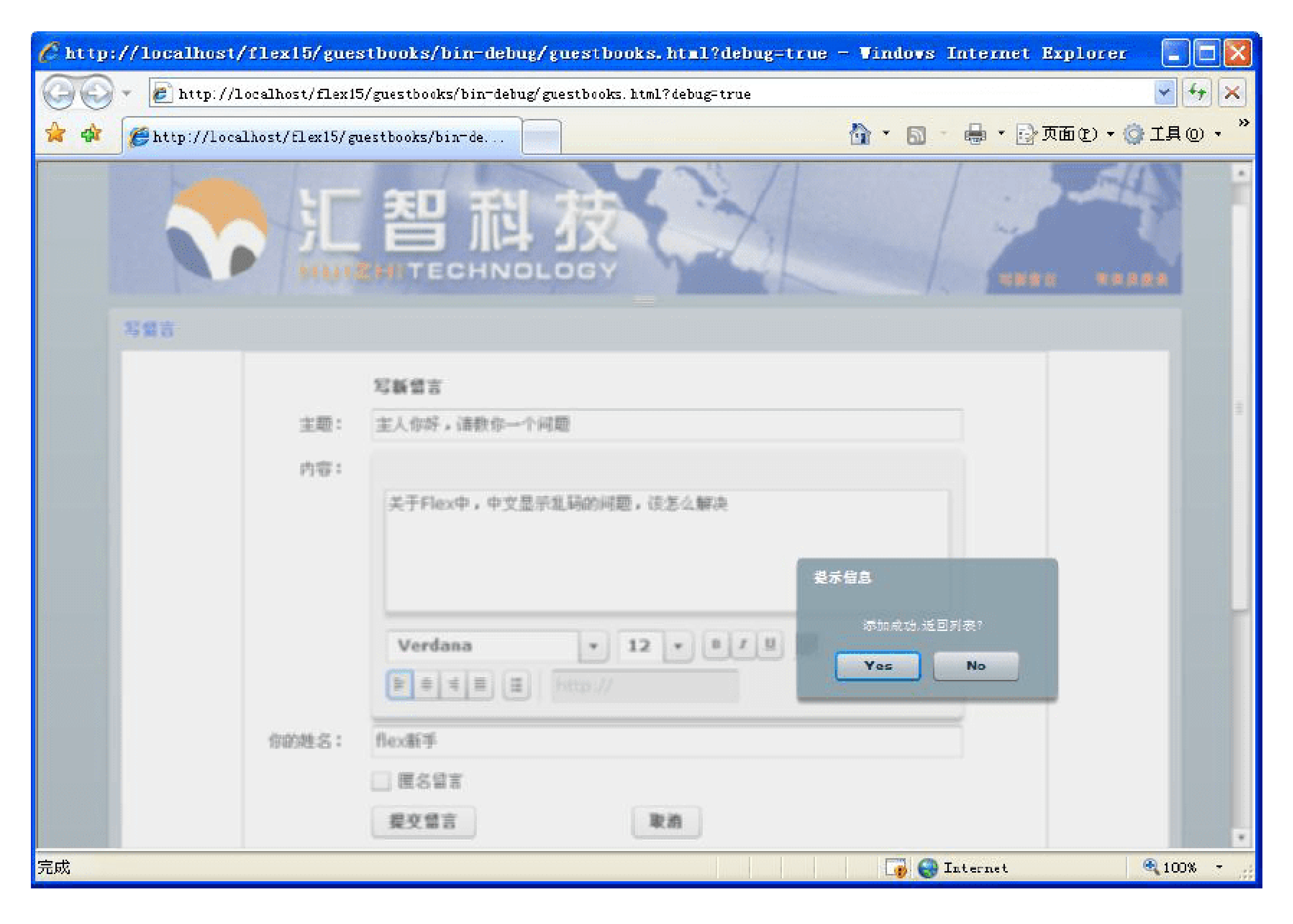


图 15-8　添加留言



图 15-9　添加留言后的留言列表

代码 15.14　查看留言详细信息界面

```
<mx:Canvas id="select">
    <mx:Panel width="800" layout="horizontal" title="查看详细信息" fontSize=
    "12" color="#263EF3" height="450" horizontalAlign="center" verticalAlign=
    "top">
```

```
            <mx:Form width="582" height="278" borderStyle="solid">
                <mx:FormHeading label="查看详细信息" width="100%" color="#000000"/>
                <mx:FormItem label="留言人: "  color="#000000" width="543">
                <mx:Text id="t_name" width="325" text="{indexname}" />
                </mx:FormItem>
                <mx:FormItem label="主题: "  color="#000000" width="543">
                <mx:Text id="t_theme" width="325" text="{indextheme}"/>
                </mx:FormItem>
                <mx:FormItem label="内容: "  color="#000000" width="543">
                    <mx:TextArea id="t_content"  width="326"  height="100" text=
                    "{indexcontent}" editable="false"/>
                </mx:FormItem>
                <mx:FormItem  color="#000000" width="542">
                  <mx:HBox>
                    <mx:Button id="wirte" label="回复此留言" />
                    <mx:Spacer width="100" />
                    <mx:Button id="exit" label="取消" click="exitto()" />
                  </mx:HBox>
                </mx:FormItem>
            </mx:Form>
        </mx:Panel>
    </mx:Canvas>
```

在代码 15.14 中，使用 Text 组件和 TextArea 组件显示详细的留言信息。实现该页面的具体功能代码如代码 15.15 所示。

代码 15.15　获取留言详细信息

```
    //获取留言详细信息
    public function selectall():void{
        if (dgData.selectedItem){
        indexname=dgData.selectedItem.userName;
        indextheme=dgData.selectedItem.Theme;
        indexcontent=dgData.selectedItem.content;
        v1.selectedChild=select;
        }
        else{
        Alert.show("请选择一项留言! ","提示信息");
        }
    }
```

在代码 15.15 中，使用 if…else 语句判断用户是否选择了留言，如果是，则获取该数据行的留言人姓名、主题、内容等信息，并在相应的组件中显示出来；如果否，则提示用户“请选择一项留言！”。查看留言详细信息界面的具体显示效果如图 15-10 所示。由于篇幅问题，这里并没有设计回复留言模块。



图 15-10　查看留言详细信息

5. 管理员登录

单击顶部的【管理员登录】命令，就可以进入管理员登录界面。该界面非常简单，具体的布局代码如代码 15.16 所示。

代码 15.16　管理员登录界面

```
<mx:Canvas id="adminlogin" label="adminlogin" width="100%" height="100%">
    <mx:Panel width="385" height="214" title="管理员登录" x="194.5" y="10"
    fontSize="12" layout="absolute">
      <mx:Label x="69" y="28" text="用户名: " width="67"/>
      <mx:Label x="69" y="78" text="密  码: " width="67"/>
      <mx:TextInput id="txt1" x="127" y="26"/>
      <mx:TextInput id="txt2" x="127" y="76" displayAsPassword="true"/>
      <mx:Button x="95" y="130" label="登录" click="check.send();"/>
      <mx:Button x="202" y="130" label="取消" click="exitto()"/>
    </mx:Panel>
</mx:Canvas>
```

在代码 15.16 中，设置【取消】按钮的 click 事件调用函数 exitto()，返回留言列表。设置【登录】按钮的 click 事件调用 HTTPService 对象的 send()方法，发送 HTTPService 请求。在这里创建了 id 为 check 的 HTTPService 对象，具体代码如代码 15.17 所示。

代码 15.17　用于验证登录的 HTTPService 对象

```
<mx:HTTPService id="check" url="../dataBase/Default.aspx" method="GET" result=
"resultLoginhandle(event)" fault=" faultHandler(event)">
      <mx:request>
```

```
        <name>{txt1.text}</name>
        <pwd>{txt2.text}</pwd>
      </mx:request>
    </mx:HTTPService>
```


在 HTTPService 对象中，设置 url 属性，并声明了结果处理函数为 resultLoginhandle()，错误处理函数为 faultHandler()。函数 faultHandler()在前面已经多次用到，这里就不再重复。下面给出了结果处理函数 resultLoginhandle()的具体代码，如代码 15.18 所示。

代码 15.18　结果处理函数：resultLoginhandle(event:ResultEvent)

```
//登录事件结果处理函数
public function resultLoginhandle(event:ResultEvent):void{
var returnValue:String=check.lastResult.result.state;
    if(returnValue=="ok"){
      username.text=txt1.text+",欢迎回来！"
      exitto();
      currentState="admin";
    }
    else{
   Alert.show("您的登录失败了","提示信息",Alert.OK,this,null,null,Alert.YES);
    }
}
```

在结果处理函数中，首先获取服务器端返回结果，使用 if...else 语句进行判断，如果登录成功则显示管理员账号，调用函数返回留言列表界面，并且设置当前状态为 admin，即显示管理员状态。具体的登录界面和登录成功后后界面如图 15-11 和图 15-12 所示。



图 15-11　管理员登录



图 15-12　管理员登录成功后的界面

在管理员界面，管理员不可以新增留言，只可以查看留言详细信息。但是管理员可以进行删除留言操作。实现该功能的具体代码如代码 15.19 所示。

代码 15.19　删除留言

```
/**
 * 在列表中单击【删除】按钮后执行
 * 弹出一个提示对话框确认操作，事件转到 deleteClickHandler 处理
 */
private function deleteItem():void{
    if (dgData.selectedItem)
    {
        Alert.show("确定要删除选定的记录?",
        "删除提示", 3, this, deleteClickHandler);
    }
}
    private function deleteClickHandler(event:CloseEvent):void{
        objHTPS.disconnect();
        //当前选择所在行的索引
        var indexForDelete:Number = dgData.selectedIndex.valueOf();
        //var currItem:XML=dgData.selectedItem;//获取当前选择行的数据
        var indexID:String=dgData.selectedItem.id;//获取选择的 id 值
        if (event.detail == Alert.YES){
            objHTPS.url="../dataBase/save_mdb.aspx";
            objHTPS.method="GET";
            objHTPS.addEventListener(ResultEvent.RESULT,ShowMessageList);
```

```
                objHTPS.addEventListener(FaultEvent.FAULT,faultHandler);
                var oArgs:Object = {};
                oArgs["DelActionID"]=indexID;
                oArgs["rndNum"]=Math.random();
                objHTPS.send(oArgs);
                Alert.show("删除成功");
        }
    }
```

在代码 15.19 中，当管理员单击【删除】按钮后，判断当前选择的留言 id，并创建 HTTPService 对象，发送 HTTPService 请求，把留言 id 作为参数传递给服务器端进行处理，删除该留言信息。

管理员删除留言的具体效果如图 15-13 所示。

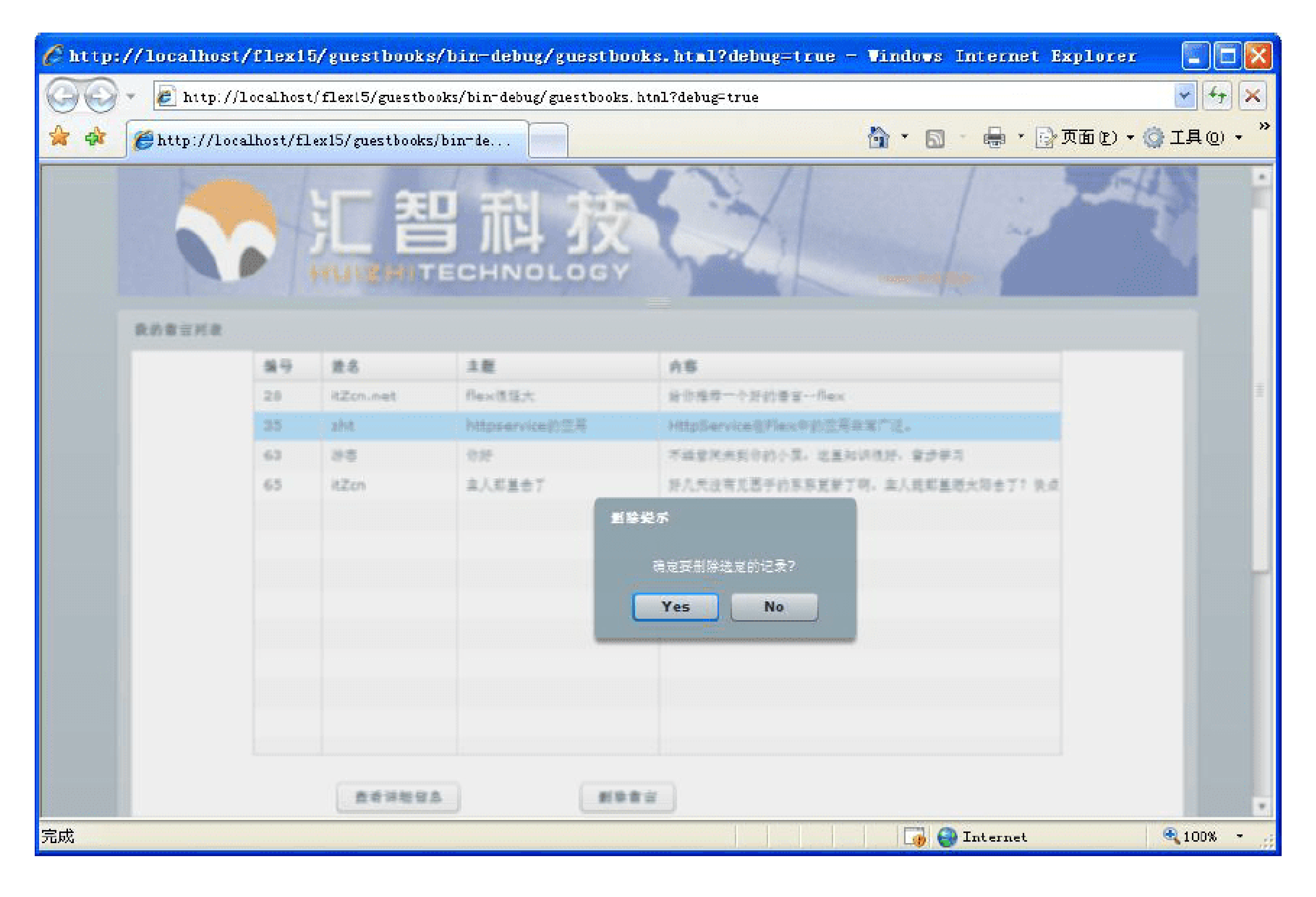


图 15-13　删除留言

15.4　使用 WebService 与服务器端交互

WebService 是一种在互联网中提供服务的技术。WebService 技术标准由各大软件开发商制定，主要解决了不同开发语言间的沟通问题。WebService 具有通用性，不论用何种语言开发的 WebService 服务，调用的结果都一致。这是因为 WebService 有自身的标准，与开发语言无关。用户可使用几乎任何语言调用 WebService 服务，只要能找到 WebService 服务并且传递正确的参数。

WebService 技术在国内外已有较广泛的应用。例如，每日的天气情况、股票走势等都是

免费的 WebService 服务。Flex 应用程序中使用<mx:WebService>组件可方便地调用 WebService 服务。

1. WebService 类库

在 Flex 3.0 中，mx.rpc.soap 下提供了一个 WebService 类。使用 WebService 对象与服务器端通信与使用 HTTPService 非常相似，也是使用 HTTP 协议来进行通信，但是其运作原理却截然不同。WebService 类常用的属性与方法如表 15-4 所示。

表 15-4　WebService 类常用的属性与方法

名称	类别	说明
description	属性	String 类型，针对当前活动端口服务的描述
destination	属性	String 类型，当前 Web 服务的接受方
endPointURL	属性	String 类型，当前 Web 服务的位置
headers	属性	Array 类型，只读属性。注册在当前 WebService 上的一组 SOAPHeaders 数组
makeObjectBindable	属性	Boolean 类型，如果为 true，返回的不知名对象强制被绑定到 Object
port	属性	String 类型，WebService 使用的端口
rootURL	属性	String 类型，指明根 URL，使用的 WebService 统一使用该 URL
service	属性	String 类型，指明当前 WebService 当中的 wsdl 文档指明的服务
useProxy	属性	Boolean 类型，指明是否使用 Flex proxy 服务，如果使用该服务则需要在 LCDS 当中描述 WebService
wsdl	属性	String 类型，WebService 的 wsdl 文档的位置
WebService()	方法	构造方法，参数为 destination，是 String 类型，默认值为 null，rootURL 是 String 类型，默认值为 null。返回一个新的 WebService
addHeader()	方法	参数为 head，是 SOAPHeaders 类型。没有返回值
canLoadWSDL()	方法	无参数，返回值是 Boolean 类型。是否可以加载 WSDL 文档
clearHeader()	方法	无参数，无返回值。清除所有的 hearders
loadWSDL()	方法	参数为 uri，是 String 类型，默认值为 null，没有返回值。加载 WSDL 文档

与 HTTPService 相比，WebService 多了一个 operation 对象，该对象表示要调用的方法，name 属性就是方法名。在 operation 对象中，可以包含 request 对象，指定要传递的参数。

2. 使用<mx:WebService>组件

与使用 HTTPService 一样，在 Flex 中用户通常会使用<mx:WebService>组件的形式来对 WebService 进行访问。<mx:WebService>组件在 mx.rpc.soap.mxml 包下，该组件实例代码如代码 15.20 所示。

代码 15.20　<mx:WebService>组件实例

```
<mx:WebService id="WeatherSrv" wsdl="http://localhost/WeatherSrv/Service.
asmx?wsdl"
    showBusyCursor="true" useProxy="false">
```

```
        <!--调用 24 小时的天气预报-->
        <mx:operation name="GetWeather" result="resultFlexWeather(event)"> <!--
        这里对应 WebService 的方法名，注意必须和 WebService 的方法同名-->
           <mx:request> <!--接收方法参数-->
               <p_strCityCode><!--传入的城市名称，这里的参数必须和 WebService 里的方法参
               数同名-->
                {selectedItem.data}
               </p_strCityCode>
           </mx:request>
       </mx:operation>
    </mx:WebService>
```

在代码 15.20 中，创建了一个用于查询城市天气的 WebService 对象，设置其 id、wsdl、showBusyCursor、useProxy 等属性。在<mx:operation>中设置其 name 属性为 GetWeather，并指明了结果处理函数为 resultFlexWeather(event)。在<mx:request>中创建参数，这里将城市作为参数进行传递。

创建完成后，在程序中就可以使用如下语句发送 WebService 请求。

```
WeatherSrv.GetWeather.send()
```

接下来需要在 Flex 程序中创建结果处理函数，具体代码如下所示。

```
private function resultFlexWeather(event:ResultEvent):void{
    //这里是具体的结果处理函数
    var arr:Array = event.result.toString().split(";");
    lblWeather.text = arr[0];
    lblTmp.text = arr[1];
    lblWind.text = arr[2];
    lblUR.text = arr[3];
    lblAir.text = arr[4];
}
```

在上述代码中，创建结果处理函数 resultFlexWeather()。在函数中，获取返回结果，并对结果进行分割，最后将分割后的数据分别在不同的组件上显示出来。

15.5　WebService 应用实例

域名查询是网站非常常见的功能之一，可以方便地查询用户需要注册的域名是否已经被注册。那么在本次实例中，将通过 WebService 创建一个域名查询的实例。

15.5.1　编写服务器端程序

前面讲到，服务器端代码与 Flex 程序代码相互独立，在 Flex 中只需要提供相应的访问接

口就可以与服务器端代码交互。这里首先设计服务器端程序。具体步骤如下所示。

（1）打开 Microsoft Visual Studio 2008，选择【文件】|【新建】|【网站】命令，弹出【新建网站】对话框，选择【ASP.NET Web 服务】模板，如图 15-14 所示。

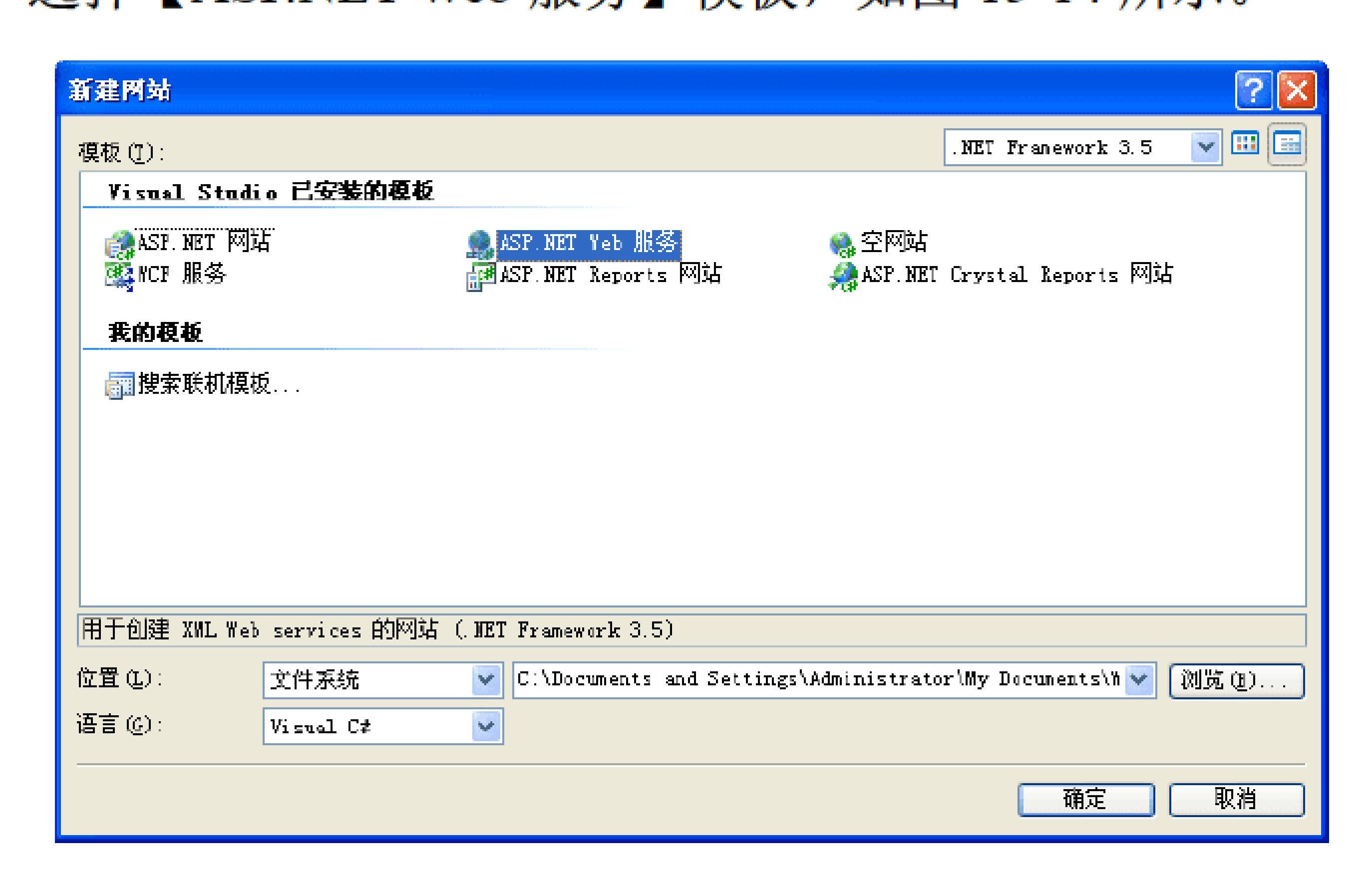


图 15-14　创建 ASP.NET Web 服务

（2）单击【浏览】按钮，选择项目路径。选择完成后单击【确定】按钮完成创建。

（3）编写 WebService。在新建“ASP.NET Web 服务”项目后会自动生成一个名为 Service.asmx 的文件。.asmx 格式文件即为 WebService 文件。按 F7 快捷键，打开 Service.asmx 的后台代码 Service.cs，编写 WebService 代码。具体如代码 15.21 所示。

代码 15.21　域名查询的服务器端代码

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Xml;

/// <summary>
///ScriptManger 使用的 Web 服务
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
//若要允许使用 ASP.NET Ajax 从脚本中调用此 Web 服务，请取消对下行的注释
//[System.Web.Script.Services.ScriptService]
public class WebService : System.Web.Services.WebService {
```

```
        public WebService () {

            //如果使用设计的组件，请取消注释以下行
            //InitializeComponent();
        }
        [WebMethod]
        public string CheckWebsite(string website) {
            string xmlstr = "";
            string strurl ="http://now.net.cn/domain/domaincheck.php?query=" +
            website;
            //声明 XmlDocument 对象实例
            XmlDocument xmldoc = new XmlDocument();
            //加载 xml 文件
            xmldoc.Load(strurl);
            //获取节点值
            XmlNode xn = xmldoc.SelectSingleNode("response/result/msg");
            xmlstr += xn.InnerText;
            return xmlstr;
        }
    }
```

在代码 15.21 中，创建了一个方法 CheckWebsite()。该方法带有一个参数 website，然后从"http://now.net.cn/domain/domaincheck.php?query=" + website 获取查询结果并赋予变量 strurl。接下来使用 XmlDocument 对象的 Load()方法加载变量 strurl，并且使用 SelectSingleNode()方法获取相应节点的信息赋予变量 xmlstr。最后返回变量 xmlstr。

（4）保存代码。这样就完成了服务器端代码的设计。

15.5.2 编写 Flex 程序

服务器端程序开发完成以后，就可以来设计 Flex 界面和具体发送 WebService 请求的 Flex 端功能代码。具体步骤如下所示。

（1）首先设计应用程序窗体界面，在本实例中，在窗口中使用了一个 Panel 组件，在该容器中使用 Label、Image、TextInput、TextArea 和 Button 按钮等组件创建一个简单的域名查询模块。具体布局代码如代码 15.22 所示。

代码 15.22 域名查询界面

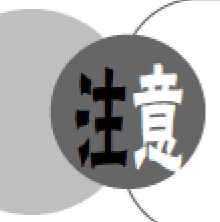
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="horizontal"
horizontalAlign="center" verticalAlign="top">
    <mx:Panel width="531" height="287" layout="absolute" borderColor=
    "#11C6ED" title="Domain Service | 域名查询" fontSize="12">
        <mx:Label x="0" y="38" text="域名查询" width="135" fontWeight="bold"
```

```
            color="#1688C3"/>
            <mx:Image x="10" y="66" width="136" height="134" source="../data/img/
            somepicture.gif"/>
            <mx:Label x="165" y="66" text="域名: www." width="79"/>
            <mx:TextInput id="txt_website" x="241" y="66" fontSize="10"/>
            <mx:Button x="409" y="64" label="查询" click="selectwhois. CheckWebsite.
            send();"/>
            <mx:TextArea x="165" y="107" width="296" height="46" editable="false"
            text="域名查询是一个非常实用的工具，可以方便地查询用户需要注册的域名是否已经被
            注册！"/>
            <mx:Text id="result" x="165" y="182" width="296" text="{res}" color=
            "red"/>
        </mx:Panel>
    </mx:Application>
```

（2）完成布局后，使用<mx:WebService>组件，创建 WebService 对象，设置其 id、wsdl、showBusyCursor 和 useProxy 等属性，设置其 wsdl 属性指向前面创建好的服务器端程序。创建<mx:operation>和<mx:request>组件，在<mx:operation>组件中定义 name 属性，并且指定返回结果的类型和结果处理函数；在<mx:request>组件中，将用户输入的域名作为参数进行传递。具体如代码 15.23 所示。

代码 15.23　查询域名的 WebService 对象

```
<mx:WebService id="selectwhois" wsdl="http://localhost/flex15/chap15/data/
WebService.asmx?wsdl" showBusyCursor="true" useProxy="false">
       <mx:operation name="CheckWebsite" resultFormat="object" result=
       "resultFlex(event) ">
          <mx:request> <!--接收方法参数-->
             <website>
              {txt_website.text}
             </website>
          </mx:request>
       </mx:operation>
</mx:WebService>
```



<mx:operation>组件的 name 属性定义了该 WebService 对象调用的方法，必须与服务器端程序中创建的方法名相同，否则将会出现错误。

（3）接下来设计具体的结果处理函数。首先引用 mx.rpc.events 的 ResultEvent 事件，然后声明变量 res，最后在结果处理函数 resultFlex()中，将返回结果赋于变量 res。具体如代码 15.24 所示。

代码 15.24　处理返回结果

```
<mx:Script>
    <![CDATA[
        import mx.rpc.events.ResultEvent;
        [Bindable]
        public var res:String;
        //结果处理函数
        private function resultFlex(event:ResultEvent):void{
          res=event.result.toString();
        }
    ]]>
</mx:Script>
```

（4）设置【查询】按钮的 click 事件调用 selectwhois.CheckWebsite.send()方法，发送 WebService 请求。并且设置 Text 组件 result 的 text 属性绑定为变量 res 的值。

（5）运行程序，具体的显示效果如图 15-15、图 15-16 所示。

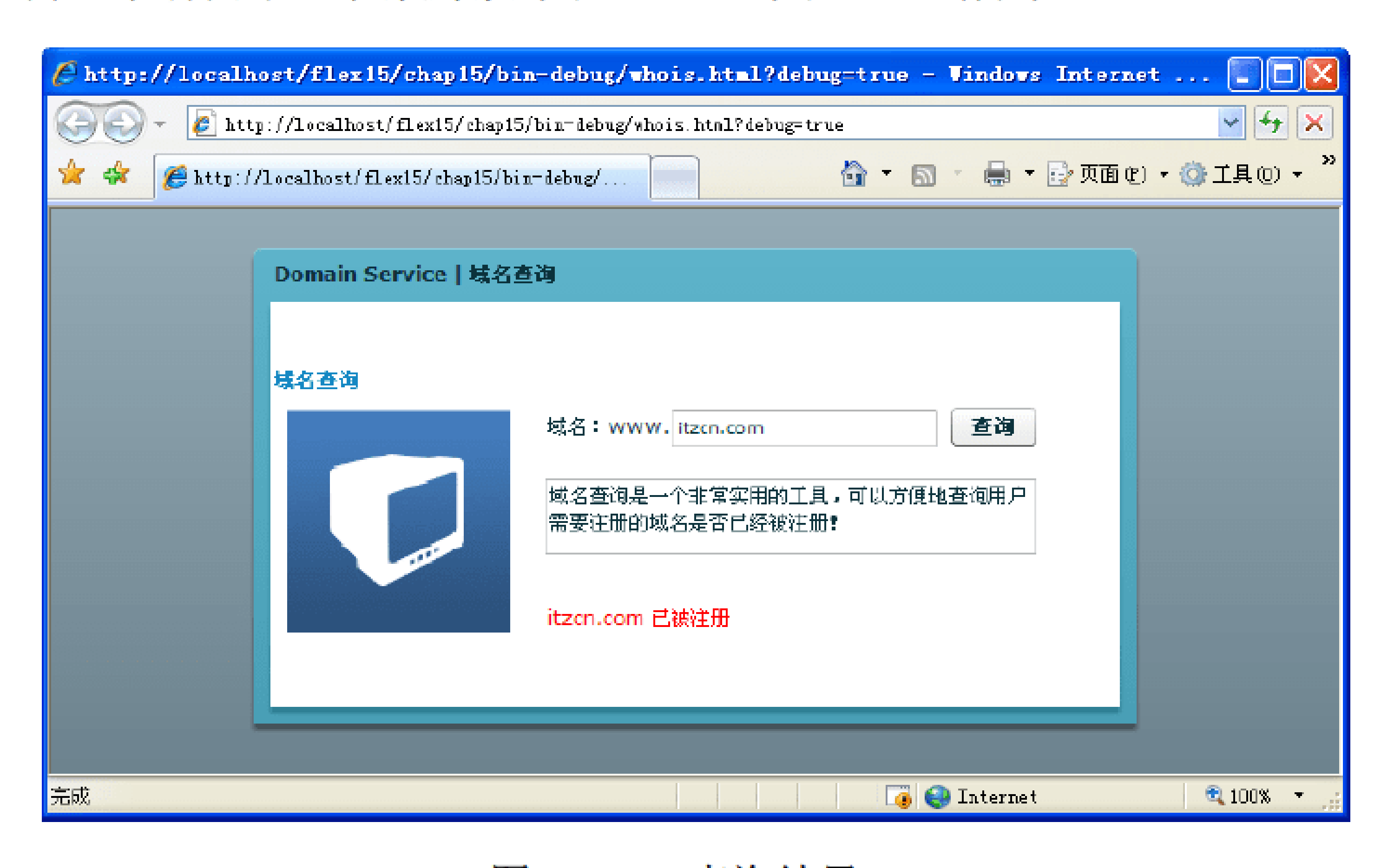


图 15-15　查询结果一

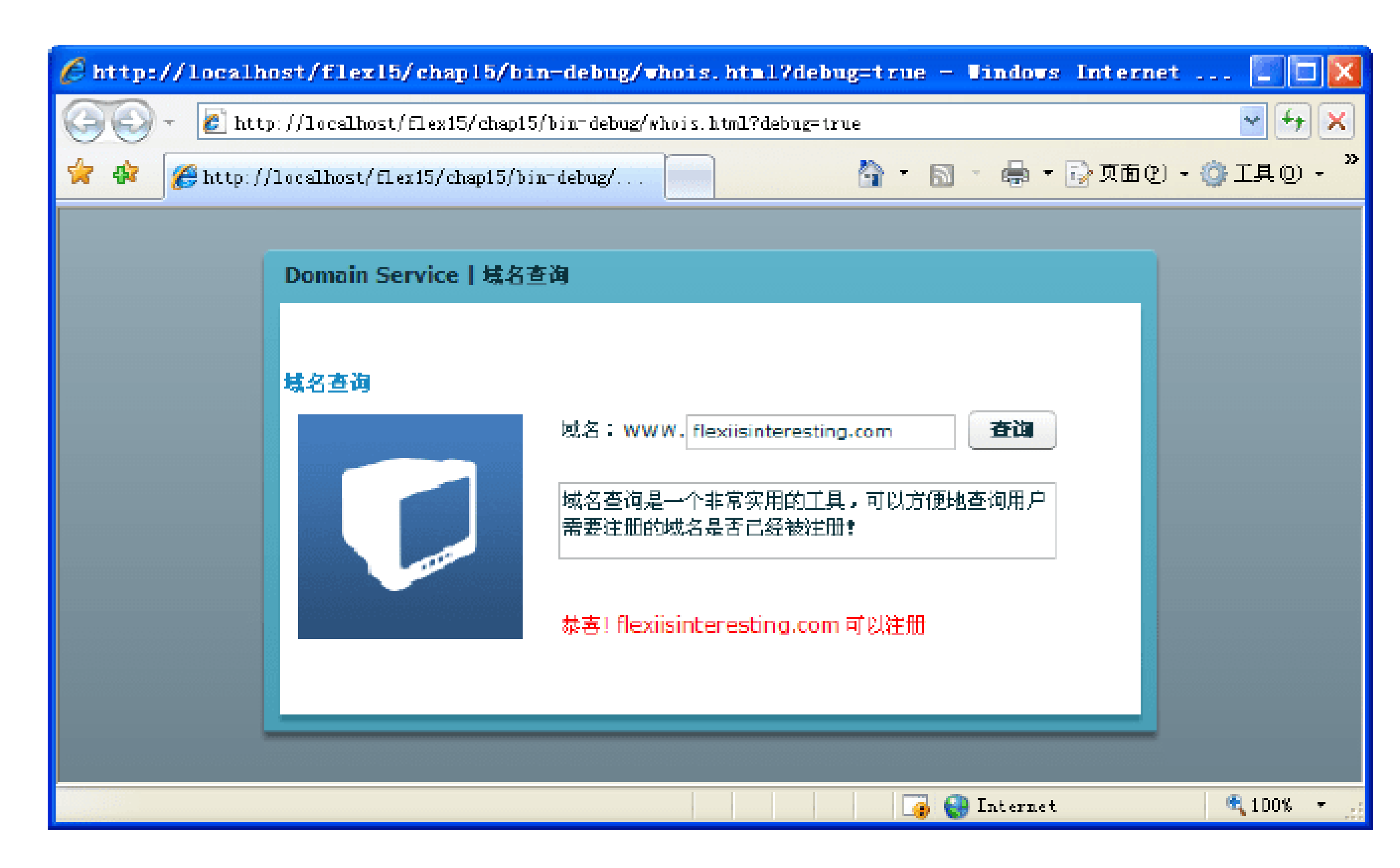
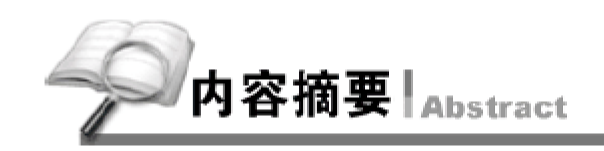


图 15-16　查询结果二

第 5 篇　综合实例篇

第16章 功能齐全的FLV播放器



FLV（Flash Video）流媒体格式是随着 Flash MX 的推出起发展起来的一种新兴的视频格式。FLV 文件体积小巧，清晰的 FLV 视频 1 分钟为 1MB 左右，一部电影为 100MB 左右，是普通视频文件体积的 1/3。再加上 CPU 占有率低、视频质量良好等特点使其在网络上盛行，目前几家著名的视频共享网站均采用 FLV 格式文件提供视频。本章将讲述如何制作一款功能齐全的 FLV 播放器。

学习目标 | Objective

- 掌握需求分析的方法
- 熟悉 FLV 播放器的结构
- 理解 XML 文件作为数据源的优势
- 掌握主题的设计方法
- 掌握色彩矩阵的应用
- 熟悉如何创建自定义事件类
- 掌握如何应用自定义事件类
- 掌握自定义组件的创建方法
- 掌握自定义组件如何与主程序交互
- 掌握事件监听过程

16.1 系统概述

做任何事情，准备工作都是必须的，开发一个系统更是如此，所以才会出现很多种软件工程的开发模型，如组件对象模型。在开始实际编码之前，首先介绍一下系统的基本情况和需求，实现时涉及的知识和其他相关内容，便于读者更好地理解系统的结构。

本系统使用了外部 XML 文件作为数据源，选择 Flex Builder 3 作为开发软件，实现在线播放 FLV 视频文件。

16.1.1 需求分析

需求分析指的就是一个和客户交流的活动，是正确引导客户能够将自己的实际需求用较为适当的技术语言进行表达（或者由相关技术人员帮助表达）以明确项目目的的过程。这个过程也同时包含了对要建立的系统基本功能、模块的确立和策划活动。

所以，在设计播放器之前应该注意：播放器设计时的可增长性功能需求、安全性及可靠性的要求、播放器的运行环境、页面总体风格以及美工效果、主程序和用户组件数量、内容管理及录入任务的分配和各种页面特殊效果等因素。

总之，在拿到项目后应分成许多不同步骤来分析和实现，它们包括以下几方面。

- 整体规划整个系统，按要求规划好每一个子系统。
- 开发小组内部进行细分，根据工程的难易，每个人分配一个子系统或者多个子系统或者多个人分配一个子系统。
- 根据要求的需要，每一个成员写出自己的模块相关文件，具体的目录和文件名。再统一定义程序整体的风格：背景颜色、图片、组件及背景图片的属性，程序的 Logo 及其大小等。
- 每个设计人员按照规划将程序所涉及的所有模块页面做出来。
- 开始分析程序的每一个子系统的数据结构，首先要规划出数据库结构，然后建立数据表。
- 根据上面的数据表，建立数据库和相应的数据表，提交实现文档。
- 组内的每个开发成员开始编写子系统代码，框架设计，模块设计，评审最后的编码。
- 由项目开发组组长或者负责人开始合并各成员的子系统，并检测程序的完整和健壮性。
- 编写用户手册。

下面以“FLV 播放器”系统为例介绍一下需要的功能。

- **FLV 播放显示器**

这是该系统的主体，负责播放当前的 FLV 文件，将此视频显示出来。为了实现能在网络中的顺畅播放，要求播放显示器具有缓冲视频的功能。

- **播放控制器**

提供与用户交互的功能，包括播放与暂停、设置视频效果及全屏等。此功能增强了播放器的可操作性，也提高了用户的兴趣。

- **播放列表**

展示给用户可以播放的文件列表，用户可以通过选择播放喜欢的视频。此功能增加了播放器的使用范围，不仅仅播放单一文件。

16.1.2 结构设计

FLV 播放器的主要作用是，通过加载外部 XML 文件，分析出其中的视频文件地址，将所有视频资料以播放列表的形式展示出来。接受用户对播放列表中视频文件的选择，并将选择

的视频文件播放和显示出来。接受用户对播放的控制，包括改变播放、暂停和重播。接受用户对视频的控制，包括调整亮度、对比度、饱和度、反转度和大小等。这样在系统中需要处理的对象有播放列表、视频显示器和控制器等。其功能主要包括显示播放列表、处理用户选择、显示视频文件、调节视频效果和大小等。

在 Flex Builder 3 中，提供了许多组件。这些组件都集成了大量的功能，例如本实例中用到的 VideoDisplay 和 TileList 组件。其中，VideoDisplay 组件用来显示视频，TileList 组件用来显示播放列表。使用起来非常容易，不需要考虑其中的原理，只需要设置相关属性。

当然，这些组件只是提供某一方面较全的功能。在程序开发中，开发者还要对其扩展，开发自己的组件，在程序中调用自己的组件。通过编写实现某一具体功能的组件，可以在主程序文件中调用，利用简洁的语句，就可以实现想要的效果。这样既方便阅读，又使得程序开发简单化。表 16-1 中列出了系统需要使用的文件列表及其完成的功能描述。

表 16-1　系统文件列表

类别	页面	功能
文件	chap16.mxml	主程序页面
	myXML\playlist.xml	播放列表文件
	myTheme\style.css	自定义主题 CSS 文件
	myEvents\PlayerEvent.as	对视频进行控制的事件处理类
	myEvents\PlayListEvent.as	对播放列表进行选择的事件处理类
	myEvents\SettingEvent.as	对视频效果进行调节的事件处理类
	myComponents\Player.mxml	视频显示和控制器
	myComponents\PlayList.mxml	视频播放列表
	myComponents\PlayListBox.mxml	播放列表的项
	myComponents\Setting.mxml	视频调节窗口
	myClass\ColorMatrix.as	对色彩进行调整的色彩矩阵类
	css\Player.css	播放器界面效果的 CSS 文件
	css\Setting.css	调节器界面效果的 CSS 文件
文件夹	myXML	包含系统用到的播放列表文件
	myTheme	包含系统的自定义主题用到的文件
	myEvents	事件处理类文件
	myComponents	用户自定义组件文件
	myClass	自定义类文件
	images	图片文件
	flv	FLV 文件

在通过表 16-1 对 FLV 播放器系统进行结构分析之后，得到了如图 16-1 所示的结构及各模块的功能图，这里仅展示了主要的功能及文件描述。

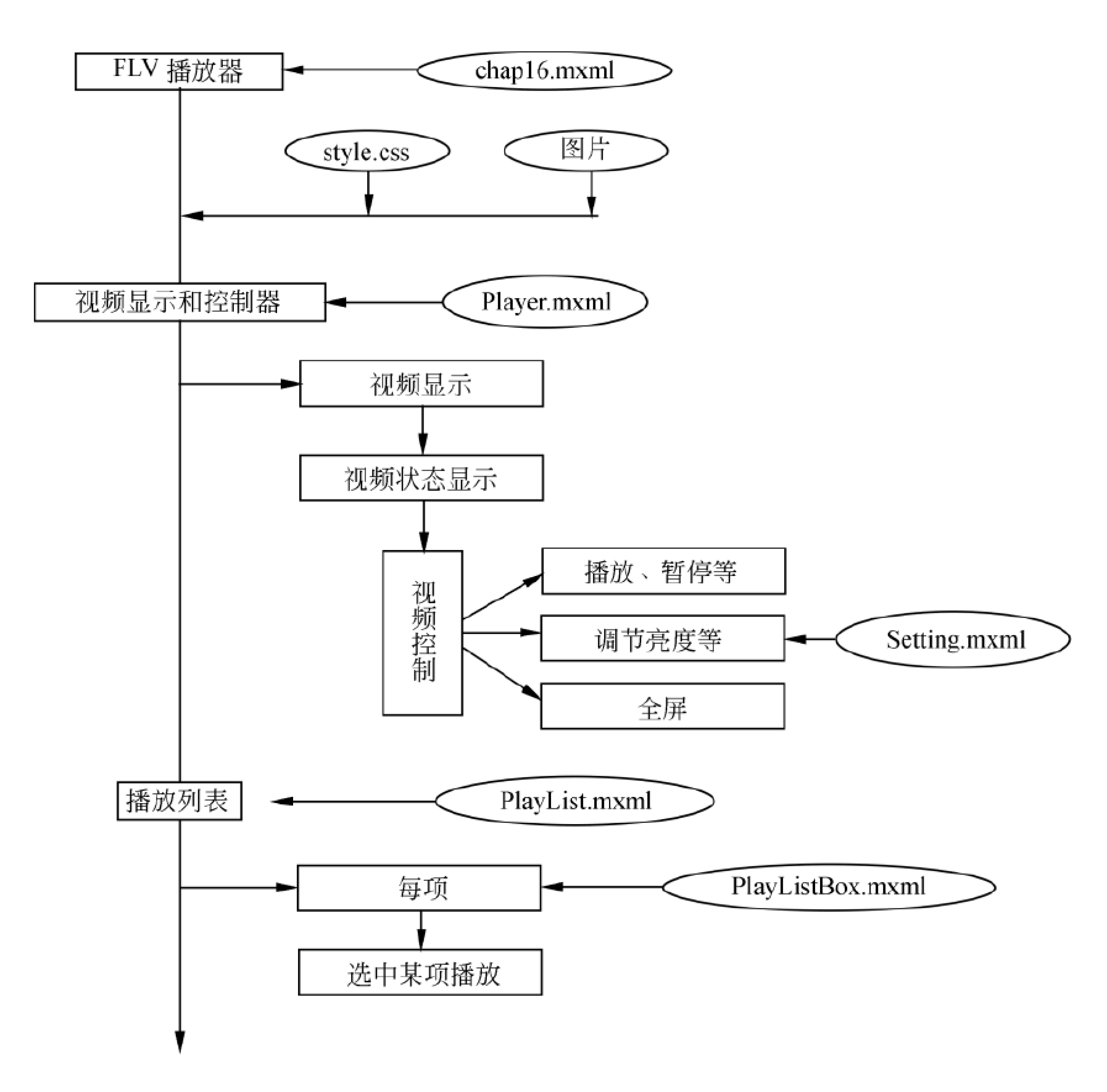


图 16-1　结构功能图

16.2　数据源、主题设计和色彩矩阵类的创建

通过上节的系统概述，明确了系统的功能需求、系统结构的设计。本节重点介绍系统中数据源、主题的设计和自定义类的创建，包括设计数据源文件及其格式以及主题的相关文件，另外包括自定义色彩矩阵类的创建。这是系统的必备基础，本系统采用 XML 文件作为数据源文件。

16.2.1　数据源文件及其格式设计

采用何种形式储存数据，是系统开发中重点考虑的内容。开发者常常会碰到需要处理以各种格式保存或者传输数据的情况，可以将长期保存的数据储存成文件或数据库，当需要时调用。选择何种形式，应该根据系统的要求，充分考虑安全性、保密性、易检索性等方面。每一种格式都需要对应的解析器，这一缺点减缓了开发进度，而且可能会导致错误的发生。

本系统采用 XML 文件形式，原因在于 Flex Builder 3 对 XML 的强大支持，操作 XML 文

件非常容易，并且 XML 文件本身也有很多优势。

XML 是一种简单的数据存储语言，使用一系列简单的标记描述数据，而这些标记可以用方便的方式建立，虽然 XML 比二进制数据要占用更多的空间，但 XML 极其简单、易于掌握和使用。

XML 与 Access、Oracle 和 SQL Server 等数据库不同，数据库提供了更强有力的数据存储和分析能力，如数据索引、排序、查找、相关一致性等，XML 仅仅是展示数据。事实上 XML 与其他数据表现形式最大的不同是：它极其简单。这是一个看上去有点细微的优点，但正是这点使 XML 与众不同。

XML 的简单使其易于在任何应用程序中读写数据，这使 XML 很快成为数据交换的唯一公共语言。虽然不同的应用软件也支持其他的数据交换格式，但不久之后它们都将支持 XML，那就意味着程序可以更容易地与 Windows、Mac OS、Linux 以及其他平台下产生的信息结合，然后可以很容易加载 XML 数据到程序中并分析它，最终以 XML 格式输出结果。

XML 使用标签来表示数据。标签由包围在一个小于号（<）和一个大于号（>）之间的文本组成，例如<tag>。起始标签（start tag）表示一个特定区域的开始，例如<start>；结束标签（end tag）定义了一个区域的结束。SGML 还定义了标签的特性（Attribute），它们是定义在小于号和大于号之间的值，例如<img src="picture.jpg">中的 src 特性。

本系统中，用到的数据主要是播放列表，包括了视频文件地址、视频名称、抓图和描述等。播放列表文件（playlist.xml）的内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<Videos>
    <video url="flv/01.flv" name="请您欣赏优美的自然风光" pic="images/01.jpg" de-
    sciption="在我国有很多优美的自然风光，大家有时间的话可以去游览一下，比如该片中的地
    方就是个好地方。"/>
    <video url="flv/02.flv" name="未来的生活" pic="images/02.jpg"desciption="
    您知道未来的生活是什么样子吗？该视频中给您展示了一种未来生活，该生活具有现代化的一切
    特征。"/>
    <video url="flv/03.flv" name="学累了就去这里逛逛" pic="images/03.jpg"descip
    tion="在我国有很多优美的自然风光，大家有时间的话可以去游览一下，比如该片中的地方就是
    个好地方。"/>
</Videos>
```

16.2.2　程序主题设计

每一个程序员在开发程序中，都想使自己的程序界面具有某种特色。如果每个页面都去设置，那样工作量将会非常庞大，特别是在 Flex Builder 中，对 CSS 样式的调整需要相当精细。

开发自己的主题，将是明智之举。这样该系统的所有页面将统一风格，不需要再单独设置，除非有特殊要求。主题的相关知识已经在第 13.3.4 小节中介绍，读者可以参考学习。

首先新建一个 myTheme 文件夹，用来存放主题文件。在此文件夹中新建一个名为 style.css 的样式文件，此文件中定义了程序中用到的组件样式。下面介绍几种组件样式的定义。先来

看影响整个场景的样式定义，主要是 global 的字体颜色和大小的定义，Application 背景的设置，代码如下所示。

```
global
{
    color: #FFFFFF;
    fontSize: 14;
}
Application
{
    backgroundColor: #000000;
    backgroundImage: Embed(source="bg.jpg");
}
```

然后是 TileList 组件的样式定义，包括背景颜色、边角圆滑度等。本实例中，播放列表使用的是该组件。该组件的样式定义代码如下所示。

```
TileList
{
    backgroundColor: #000000;
    cornerRadius: 5;
    selectionColor: #BC8500;
    textRollOverColor: #888888;
    textSelectedColor: #F60000;
    rollOverColor: #8E0000;
}
```

再来看 Panel 和 TitleWindow 组件的样式定义，主要包括边框样式、头部样式、背景颜色和关闭按钮等样式项目。本实例中，播放显示器和调节器将受影响。样式定义代码如下所示。

```
Panel {
    borderColor: #cccccc;
    borderAlpha: 1;
    borderThicknessLeft: 1;
    borderThicknessTop: 0;
    borderThicknessBottom: 1;
    borderThicknessRight: 1;
    roundedBottomCorners: true;
    cornerRadius: 6;
    headerHeight: 22;
    backgroundAlpha: 1;
    highlightAlphas: 0.24, 0;
    headerColors: #000000, #333333;
    backgroundColor: #000000;
    dropShadowEnabled: false;
    titleStyleName: "mypanelTitle";
```

```
        horizontalAlign: center;
        paddingLeft: 0;
        paddingRight: 0;
        paddingTop: 0;
        paddingBottom: 0;
        verticalAlign: middle;
    }
    .mypanelTitle {
        color: #ffffff;
        fontSize: 14;
    }
    TitleWindow
    {
        closeButtonSkin: Embed(source="titlewindow_close.png");
        closeButtonUpSkin: Embed(source="titlewindow close.png");
        closeButtonOverSkin: Embed(source="titlewindow close over.png");
        closeButtonDownSkin: Embed(source="titlewindow_close.png");
    }
```

接着来看 Hslider、ProgressBar 和 ToolTip 组件样式的定义，主要包括滑块的图片、滑动轨道的图片、进度条的颜色、工具提示的颜色等。本实例中，视频的播放状态用到了这 3 个组件。样式定义代码如下所示。

```
    HSlider {
        labelOffset: 10;
        thumbOffset: 0;
        dataTipOffset: 10;
        tickOffset: 3;
        tickLength: 4;
        tickThickness: 2;
        tickColor: #cccccc;
        showTrackHighlight: true;
        borderColor: #333333;
        themeColor: #FFAB00;
        thumbSkin: Embed(source="scroll bit.png");
        showDataTip:false;
        useHandCursor:true;
        trackSkin: Embed(source="slider_track.png");
    }
    ProgressBar {
        barColor: #cccccc;
        trackColors: #010101, #454545;
        paddingLeft: 0;
        paddingRight: 0;
        textIndent: 0;
        trackHeight: 4;
```

```
        verticalGap: 0;
        themeColor: #3A3A3A;
        labelWidth: 0;
    }
    ToolTip {
        backgroundAlpha: 0.36;
        cornerRadius: 1;
        backgroundColor: #000000;
        color: #ffffff;
        fontSize: 10;
    }
```

创建完成之后，将图片复制到 myTheme 文件夹，通过项目属性的设置，将该主题应用到本实例项目中。设置成功后，将 TitleWindow、Hslider、Button 和 ProgressBar 组件添加到场景中，运行后，效果如图 16-2 所示。

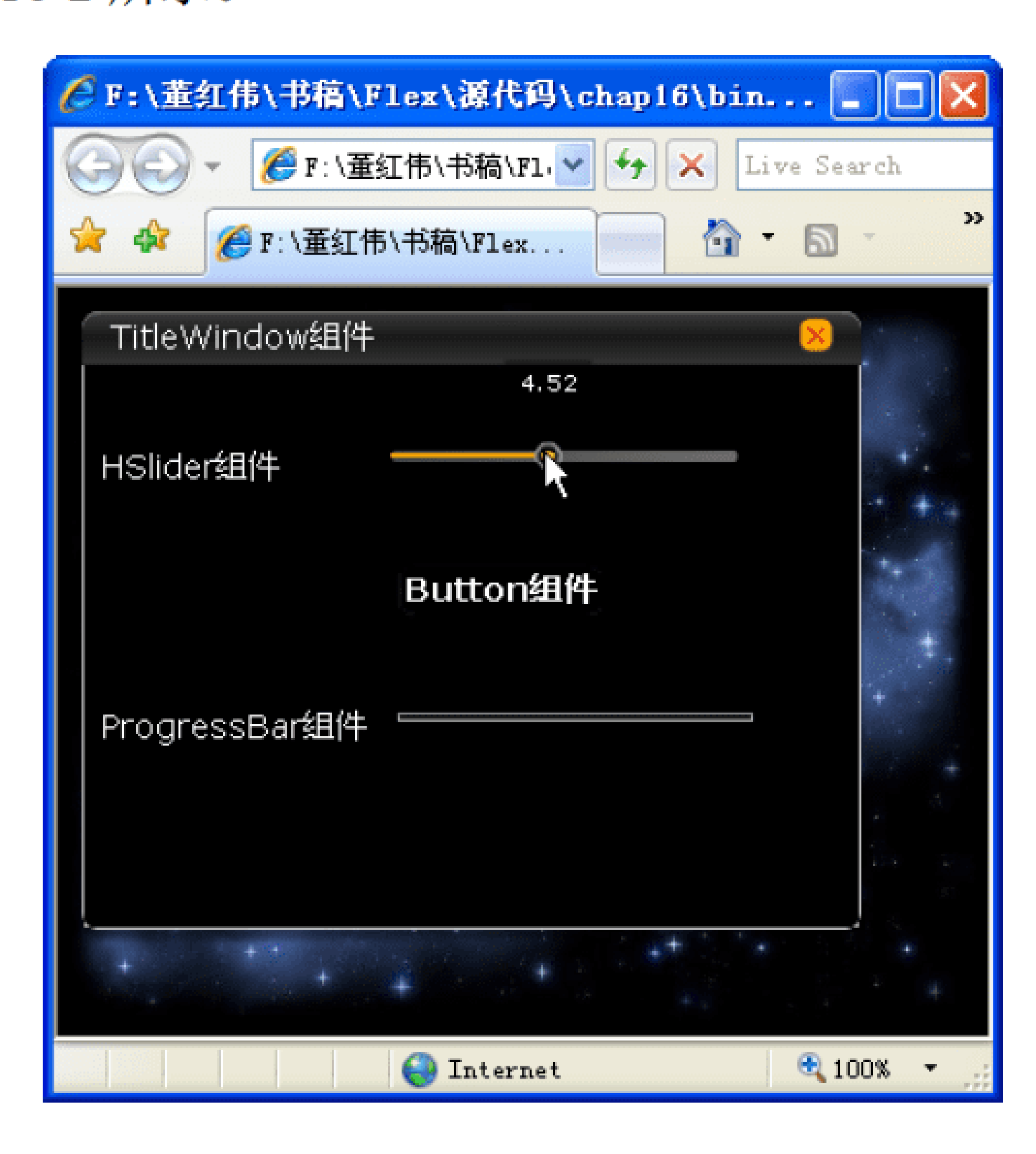


图 16-2　设置自定义主题后效果

16.2.3　色彩矩阵类的创建

在 Flex 中，可视化对象可以用矩阵方便地操作颜色，ColorMatrixFilter（flash.filters.Color MatrixFilter）在颗粒等级上提供给用户更好的控制方法。

使用 ColorMatrixFilter 类可以将 4×5 矩阵，转换成应用于图像上每个像素的 RGBA 和 Alpha 值，以生成具有一组新的 RGBA 颜色和 Alpha 值的结果。该类允许饱和度更改、色相旋转、亮度为 Alpha 以及其他各种效果。可以将滤镜应用于任何显示对象（即从 DisplayObject 类继承的对象），如 Button、Text、Video 以及 Image 对象等。

ColorMatrixFilter 将每个源像素分离成它的红色、绿色、蓝色和 Alpha 成分（可选），可以

将偏移量（介于–255 至 255 之间）添加到每个结果（矩阵每行中的第 5 项）中。滤镜将各颜色成分重新组合为单一像素，并写出结果。

本实例中创建的 ColorMatrix 类，可以创建 5×5 浮点数矩阵，增加一行虚拟位。ColorMatrix 是通过 5×5 矩阵，对图像颜色（包括 Alpha）进行的几何变换。理解和掌握 ColorMatrix 变换，能使显示对象产生千变万化的效果。下面讲述 ColorMatrix 类的创建过程及含义。

首先在 Flex 中新建一个名为 ColorMatrix.as 的类文件，该类继承于 Array 类。为该类添加 3 个常量属性，代码如下所示。

```
package myClass
{
    dynamic public class ColorMatrix extends Array {
        //计算对比度用到的常量数组
        private static const DELTA INDEX:Array = [
            0,    0.01, 0.02, 0.04, 0.05, 0.06, 0.07, 0.08, 0.1,  0.11,
            0.12, 0.14, 0.15, 0.16, 0.17, 0.18, 0.20, 0.21, 0.22, 0.24,
            0.25, 0.27, 0.28, 0.30, 0.32, 0.34, 0.36, 0.38, 0.40, 0.42,
            0.44, 0.46, 0.48, 0.5,  0.53, 0.56, 0.59, 0.62, 0.65, 0.68,
            0.71, 0.74, 0.77, 0.80, 0.83, 0.86, 0.89, 0.92, 0.95, 0.98,
            1.0,  1.06, 1.12, 1.18, 1.24, 1.30, 1.36, 1.42, 1.48, 1.54,
            1.60, 1.66, 1.72, 1.78, 1.84, 1.90, 1.96, 2.0,  2.12, 2.25,
            2.37, 2.50, 2.62, 2.75, 2.87, 3.0,  3.2,  3.4,  3.6,  3.8,
            4.0,  4.3,  4.7,  4.9,  5.0,  5.5,  6.0,  6.5,  6.8,  7.0,
            7.3,  7.5,  7.8,  8.0,  8.4,  8.7,  9.0,  9.4,  9.6,  9.8,
            10.0
        ];
        //单位矩阵常量（从左上至右下，主对角线依次为红、绿、蓝、Alpha 和虚拟位）
        private static const IDENTITY_MATRIX:Array = [
            1,0,0,0,0,
            0,1,0,0,0,
            0,0,1,0,0,
            0,0,0,1,0,
            0,0,0,0,1
        ];
        private static const LENGTH:Number = IDENTITY_MATRIX.length;
    }
}
```

然后添加私有方法 copyMatrix()，该方法将指定的矩阵复制到类对象，这里用到对数组的赋值操作。代码如下所示。

```
//私有方法
//将指定矩阵复制到本类的对象
protected function copyMatrix(p_matrix:Array):void {
    var l:Number = LENGTH;
    for (var i:uint=0;i<l;i++) {
```

```
            this[i] = p_matrix[i];
        }
    }
```

添加 multiplyMatrix()方法，该方法实现类对象与指定矩阵相乘，这是对矩阵的常用操作。在改变显示对象亮度、对比度等性质时都会用到。代码如下所示。

```
    //一个矩阵*一个矩阵
    protected function multiplyMatrix(p_matrix:Array):void {
        var col:Array = [];
        for (var i:uint=0;i<5;i++) {
            for (var j:uint=0;j<5;j++) {
                col[j] = this[j+i*5];
            }
            for (j=0;j<5;j++) {
                var val:Number=0;
                for (var k:Number=0;k<5;k++) {
                    val += p_matrix[j+k*5]*col[k];
                }
                this[j+i*5] = val;
            }
        }
    }
```

添加 cleanValue()方法，该方法用在调整显示对象的性质时，保证所调值在一定的范围，不能越界。代码如下所示。

```
    //保证值不越界。例如反转度最大为 180，其他为 100。
    protected function cleanValue(p_val:Number,p_limit:Number):Number {
        return Math.min(p_limit,Math.max(-p_limit,p_val));
    }
```

添加 fixMatrix()方法，该方法将指定的矩阵转换为 5×5 的矩阵，主要用在外部矩阵传递进来时，对外部矩阵进行处理。代码如下所示。

```
    //保证矩阵为 5×5（长度为 25）
    protected function fixMatrix(p matrix:Array=null):Array {
        if (p matrix == null) { return IDENTITY MATRIX; }
        if (p matrix is ColorMatrix) { p matrix = p matrix.slice(0); }
        if (p matrix.length < LENGTH) {
            p_matrix=p_matrix.slice(0,p_matrix.length).concat(IDENTITY_MATRIX.
            slice(p_matrix.length,LENGTH));
        } else if (p_matrix.length > LENGTH) {
            p_matrix = p_matrix.slice(0,LENGTH);
        }
        return p_matrix;
    }
```

添加构造函数，该构造函数的参数默认值为 null，通过调用 fixMatrix()和 copyMatrix()方法，创建 5×5 的矩阵。代码如下所示。

```
//构造函数:
public function ColorMatrix(p_matrix:Array=null) {
    p_matrix = fixMatrix(p_matrix);
    copyMatrix(((p_matrix.length == LENGTH) ? p_matrix : IDENTITY_MATRIX));
}
```

添加调节亮度的 adjustBrightness()方法，该方法通过改变 R、G、B 这 3 个通道的偏移量，实现亮度的调节。代码如下所示。

```
public function adjustBrightness(p val:Number):void {
    p_val = cleanValue(p_val,100);
    if (p_val == 0 || isNaN(p_val)) { return; }
    multiplyMatrix([
        1,0,0,0,p_val,
        0,1,0,0,p_val,
        0,0,1,0,p_val,
        0,0,0,1,0,
        0,0,0,0,1
    ]);
}
```

添加 adjustContrast()方法，该方法用来调节对比度。主要是通过调节 R、G、B 通道的主对角线的值，以及 R、G、B 通道的分量，达到对比度的调节，代码如下所示。

```
public function adjustContrast(p_val:Number):void {
    p_val = cleanValue(p_val,100);
    if (p_val == 0 || isNaN(p_val)) { return; }
    var x:Number;
    if (p_val<0) {
        x = 127+p_val/100*127
    } else {
        x = p_val%1;
        if (x == 0) {
            x = DELTA INDEX[p val];
        } else {
            x = DELTA_INDEX[(p_val<<0)]*(1-x)+DELTA_INDEX[(p_val<<0)+1]*x;
        }
        x = x*127+127;
    }
    multiplyMatrix([
        x/127,0,0,0,0.5*(127-x),
        0,x/127,0,0,0.5*(127-x),
        0,0,x/127,0,0.5*(127-x),
```

```
            0,0,0,1,0,
            0,0,0,0,1
        ]);
    }
```


添加 adjustSaturation()方法，该方法实现调节饱和度的功能。通过改变 R、G、B 通道上所有值，达到色彩饱和度的调节，代码如下所示。

```
    public function adjustSaturation(p_val:Number):void {
        p_val = cleanValue(p_val,100);
        if (p_val == 0 || isNaN(p_val)) { return; }
        var x:Number = 1+((p_val > 0) ? 3*p_val/100 : p_val/100);
        var lumR:Number = 0.3086;
        var lumG:Number = 0.6094;
        var lumB:Number = 0.0820;
        multiplyMatrix([
            lumR*(1-x)+x,lumG*(1-x),lumB*(1-x),0,0,
            lumR*(1-x),lumG*(1-x)+x,lumB*(1-x),0,0,
            lumR*(1-x),lumG*(1-x),lumB*(1-x)+x,0,0,
            0,0,0,1,0,
            0,0,0,0,1
        ]);
    }
```

添加 adjustHue()方法，该方法用来调节反转度。主要是通过数学运算，包括正、余弦，将现有像素反转，代码如下所示。

```
    public function adjustHue(p_val:Number):void {
        p_val = cleanValue(p_val,180)/180*Math.PI;
        if (p_val == 0 || isNaN(p_val)) { return; }
        var cosVal:Number = Math.cos(p_val);
        var sinVal:Number = Math.sin(p_val);
        var lumR:Number = 0.213;
        var lumG:Number = 0.715;
        var lumB:Number = 0.072;
        multiplyMatrix([lumR+cosVal*(1-lumR)+sinVal*(-lumR),lumG+cosVal*
        (-lumG)
    +sinVal*(-lumG),lumB+cosVal*(-lumB)+sinVal*(1-lumB),0,0,lumR+cosVal*(-lumR)
    +sinVal*(0.143),lumG+cosVal*(1-lumG)+sinVal*(0.140),lumB+cosVal*(-lumB)+sin
    Val*(-0.283),0,0,lumR+cosVal*(-lumR)+sinVal*(-(1-lumR)),lumG+cosVal*(-lumG)
    +sinVal*(lumG),lumB+cosVal*(1-lumB)+sinVal*(lumB),0,0,0,0,0,1,0,0,0,0,0,1]);
    }
```

添加 adjustColor()方法，该方法调用上文中的方法，对显示对象进行综合调节，包括亮度、对比度、饱和度和反转度。代码如下所示。

```
    public function adjustColor(p_brightness:Number,p_contrast:Number,p_satura-
```

```
tion:Number,p_hue:Number):void {
    adjustHue(p_hue);
    adjustContrast(p_contrast);
    adjustBrightness(p_brightness);
    adjustSaturation(p_saturation);
}
```


16.3　编写事件类

自定义组件和主程序的交互，要靠不同参数的事件实现。不同自定义组件中的相同组件，它们的事件参数是相同的。这样，主程序就很难辨别。为了能实现不同自定义组件传递不同的时间参数，需要自定义的事件类。该类相当于自定义组件和主程序间的桥梁。本节中讲述了本实例中自定义的 3 个事件类。

16.3.1　视频控制器事件类

视频控制器事件类继承自 Event 类，主要是在视频播放和控制器组件中，对视频播放状态和全屏的事件。该事件构造函数的参数，就是事件的参数。代码如下所示。

```
package myEvents
{
    import flash.events.Event;
    public class PlayerEvent extends Event
    {
        public function PlayerEvent(type:String)
        {
            super(type);
        }
    }
}
```

16.3.2　视频调节器事件类

视频调节器事件类，是调节器中的事件。该类有 4 个公有属性，包括了亮度、对比度、饱和度和反转度，可以从外部访问。构造函数的参数为 4 个数字，分别赋予 4 个属性，然后发送参数为 ChangeSetting 的事件到事件流。代码如下所示。

```
package myEvents
{
    import flash.events.Event;
    public class SettingEvent extends Event
```

```
    {
        public var Bright:Number;
        public var Contrast:Number;
        public var Saturation:Number;
        public var Hue:Number;
        public function SettingEvent(bright:Number,contrast:Number,saturati-
        on:Number,hue:Number)
        {
            Bright=bright;
            Contrast=contrast;
            Saturation=saturation;
            Hue=hue;
            super("ChangeSetting");
        }

    }
}
```

16.3.3 播放列表事件类

播放列表事件类，是播放列表中的事件。该类有 1 个公有属性，是一个 Object 类对象。构造函数的参数为 Object 类对象，赋予属性，然后发送参数为 ChangePlayList 的事件到事件流。代码如下所示。

```
package myEvents
{
    import flash.events.Event;
    public class PlayListEvent extends Event
    {
        public var video:Object;
        function PlayListEvent(_video:Object):void
        {
            video=_video;
            super("ChangePlayList");
        }
    }
}
```

16.4 自定义组件设计

将具有特定功能的部分创建为自定义组件，通过事件处理类与外部组件交互，这是程序对象化设计的模式。本实例采取的就是这种设计模式，与视频播放相关的功能组件保存到

Player.mxml 文件中，整个播放列表保存到 PlayList.mxml 文件中，播放列表中的项保存到 PlayListBox.mxml 文件中，调节器保存到 Setting.mxml 文件中。下面对各个自定义组件进行详细解释。

16.4.1　视频播放和控制器

视频播放和控制器是本实例中的核心，主要实现视频的加载与播放，视频状态的显示，以及对视频的操作。该组件基于 Canvas 组件创建，视频播放使用 VideoDisplay 组件，视频加载进度使用 ProgressBar 组件，视频播放进度使用 Hslider 组件，其他的视频控制使用 Button 组件。该组件文件的内容及布局代码如下所示。

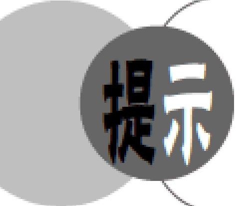
```
<?xml version="1.0" encoding="utf-8"?>
    <mx:Style source="css/Player.css">
    </mx:Style>
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"  width="400" height="5
00" creationComplete="init()">
    <mx:Panel width="100%" layout="absolute" title="{nowPlaying}" vertical-
    Align="middle" horizontalAlign="center" height="380" left="0" top="0"
    id="PlayerPanel" doubleClickEnabled="true" doubleClick="ChangeState(ev-
    ent)">
        <mx:VideoDisplay x="0" y="0" width="100%" height="100%" id="myVideo-
        Display" includeInLayout="true" cornerRadius="20" useHandCursor="t-
        rue"/>
        <mx:Canvas x="0" y="0" width="100%" height="100%" click="PlayerEventMe
        ('Pause')" id="canvasPause" buttonMode="true" mouseChildren="false"
        useHandCursor="true">
        </mx:Canvas>
        <mx:Canvas x="0" y="0" width="100%" height="100%" click="PlayerEvent
        Me('Play')" id="canvasPlay" visible="false" alpha="0.8" buttonMode=
        "true" mouseChildren="false" useHandCursor="true">
            <mx:Button  width="100" height="100" horizontalCenter="0" vert-
            icalCenter="0"  styleName="BigPlayButton"/>
        </mx:Canvas>
        <mx:Canvas x="0" y="0" width="100%" height="100%" click="PlayerEve-
        ntMe('RePlay')" id="canvasRePlay" visible="false" buttonMode="true"
        mouseChildren="false" useHandCursor="true">
            <mx:Button  width="100" height="100" horizontalCenter="0" verti-
            calCenter="0"  styleName="replay"/>
            <mx:Text text="重播" horizontalCenter="0" verticalCenter="-59"/>
        </mx:Canvas>
    </mx:Panel>
    <mx:ProgressBar width="100%" labelPlacement="top" height="1" maximum=
    "100" minimum="0" mode="event" trackHeight="5" source="{myVideoDisplay}"
    horizontalCenter="0" paddingLeft="6" paddingRight="6" id="progressbar1"
```

```
        bottom="127"/>
        <mx:HSlider width="100%" horizontalCenter="0" minimum="0" maximum="{myVi-
        deoDisplay.totalTime}" id="sliderVideoPosition" value="{myVideoDisplay.
        playheadTime}" bottom="98" buttonMode="true" useHandCursor="true">
           <mx:trackSkin>@Embed(source='../images/player_slider_track.png')</mx:
            trackSkin>
        </mx:HSlider>
        <mx:HBox width="220" height="40" horizontalCenter="0" borderThickness="1"
        borderColor="#C7C7C7" horizontalGap="20" id="hbox" bottom="48">
            <mx:Button id="btSmallPlay" buttonMode="true" mouseChildren="false"
            useHandCursor="true"/>
            <mx:Button styleName="smallsetting" click="PlayerEventMe('ShowSett-
            ing')"buttonMode="true"mouseChildren="false"useHandCursor="true"/>
            <mx:Button styleName="smallfullscreen" click="PlayerEventMe('FullS-
            creen')" id="button1" buttonMode="true" mouseChildren="false" useHand
            Cursor="true"/>
        </mx:HBox>
    </mx:Canvas>
```

上述代码中，id 为 canvasPause 的 Canvas 组件，为视频播放器上的暂停组件。由于该组件所处深度在视频播放器中最高，单击视频播放器，触发该组件的 click 事件。

同上，id 为 canvasPlay 的 Canvas 组件，为视频播放器上的播放组件。id 为 canvasRePlay 的 Canvas 组件，为视频播放器上的重播组件。

显示视频播放进度的 Hslider 组件，其 trackSkin 属性设置为一张半透明图片。这样，就可以看到视频的加载进度条了。



鼠标移到组件上方显示手的形状，除了设置 useHandCursor 为 true 外，还要设置 buttonMode 为 true。

由于该播放器要实现全屏显示功能，全屏后该视频播放和控制器的布局要发生改变。使用过 Flash 的读者，会想到在另外一帧处理全屏布局。在 Flex 中，为了实现此效果，使用 states 是最佳选择，通过创建 states，可以实现 Flash 中的帧效果。在本实例中，创建一个名为 FullScreen 的 State。在此 State 中，改变各组件为全屏样式和功能，代码如下所示。

```
    <mx:states>
        <mx:State name="FullScreen">
            <mx:SetProperty target="{PlayerPanel}" name="width"  value="100%"/>
            <mx:SetProperty target="{PlayerPanel}" name="height"  value="100
            %"/>
            <mx:SetProperty target="{hbox}" name="y"/>
            <mx:SetEventHandler target="{button1}" name="click" handler="Player
            EventMe('NormalScreen')"/>
    </mx:State>
```

```
</mx:states>
```

添加 Script 脚本，首先定义变量，包括判断视频是否在播放的 isvideoplayed，用来储存播放或暂停等操作开始前的播放状态。nowPlaying 变量储存当前播放的视频文件名，该变量绑定到 Panel 组件的 title 属性。代码如下所示。

```
<mx:Script>
    <![CDATA[
        import mx.events.VideoEvent;
        import mx.events.SliderEvent;
        import myEvents.PlayerEvent;
        //判断视频是否在播放
        [Bindable]
        public var isvideoplayed:Boolean=true;
        //当前播放的视频文件名
        [Bindable]
        public var nowPlaying:String;
    ]]>
</mx:Script>
```

添加视频播放和控制器的初始化函数 init()，该函数为场景中的对象添加事件监听器，为各种操作指定事件处理函数，以及根据 isvideoplayed 的值为播放按钮指定样式。代码如下所示。

```
private function init():void{
    this.sliderVideoPosition.addEventListener(MouseEvent.MOUSE_DOWN,Rememb
    erOldState);
    this.sliderVideoPosition.addEventListener(MouseEvent.MOUSE UP,ToThisPo-
    stion);
    this.sliderVideoPosition.addEventListener(SliderEvent.CHANGE,ChangeVid-
    eoPosition);
    this.myVideoDisplay.addEventListener(VideoEvent.COMPLETE,VideoComplete
    );
    this.btSmallPlay.styleName=isvideoplayed?"smallpause":"smallplay";
    this.btSmallPlay.addEventListener(MouseEvent.CLICK,Play_Click);
}
```

添加 RememberOldState()函数，该函数是鼠标单击播放进度条时的处理函数。该函数用来在用户改变播放进度条位置时，暂停播放的视频。这样，在鼠标松开后再做另外处理，达到改变视频播放进度的效果。如果通过其他方式实现此效果，在改变进度条位置时，有可能会出现不流畅的效果。该函数的代码如下所示。

```
private function RememberOldState(clickevent:MouseEvent):void{
    if(this.myVideoDisplay.playing){
        this.myVideoDisplay.pause();
        this.isvideoplayed=true;
```

```
        }else{
            this.isvideoplayed=false;
        }
    }
```

添加 ToThisPostion()函数，该函数是鼠标松开播放进度条时的处理函数。该函数用来在用户改变播放进度条位置时，恢复改变前的播放状态，代码如下所示。

```
    private function ToThisPostion(clickevent:MouseEvent):void{
        if(this.isvideoplayed){
            this.myVideoDisplay.play();
        }
    }
```

添加 ChangeVideoPosition()函数，该函数是改变播放进度条时的处理函数。该函数用来在用户改变播放进度条位置时，动态改变视频的播放进度，代码如下所示。

```
    private function ChangeVideoPosition(sliderevent:SliderEvent):void{
        this.myVideoDisplay.playheadTime=this.sliderVideoPosition.value;
    }
```

添加 VideoComplete()函数，该函数是当前视频播放完成后的事件处理函数。该函数将播放状态标记为非播放，然后向其父级发送参数类型为 PlayComplete 的 PlayerEvent 事件，交由其父级处理。代码如下所示。

```
    private function VideoComplete(venvet:VideoEvent):void{
        this.isvideoplayed=false;
        this.parentApplication.dispatchEvent(new PlayerEvent("PlayComplete"));
    }
```

添加 Play_Click()函数，该函数是在单击【播放】按钮时的事件处理函数。该函数根据当前播放状态，调用 PlayerEventMe()函数，传递不同的参数值。代码如下所示。

```
    private function Play Click(clickevent:MouseEvent):void{
        if(this.isvideoplayed){
            PlayerEventMe("Pause");
        }else{
            PlayerEventMe("Play");
        }
    }
```

添加 PlayerEventMe()函数，该函数将传递的参数作为 PlayerEvent 事件的参数，发送到其父级的事件流，主要实现对视频的播放状态及全屏的控制请求。代码如下所示。

```
    public function PlayerEventMe(type:String):void{
        this.parentApplication.dispatchEvent(new PlayerEvent(type));
    }
```

添加双击播放器 Panel 组件的事件处理函数 ChangeState()。该函数调用 PlayerEventMe() 函数，发送全屏或取消全屏的请求，代码如下所示。

```
private function ChangeState(doubleclick:MouseEvent):void{
    if(this.currentState==null){
        PlayerEventMe("FullScreen");
    }else{
        PlayerEventMe("NormalScreen");
    }
}
```

16.4.2　播放列表

播放列表文件相当简单，就是一个 TileList 组件，但是其 itemRenderer 属性为另外的一个组件 PlayListBox。将这两者分开编写，当需要对播放列表整体改变时，只需要对单项文件或播放列表文件进行简单修改，分析起来相当简单。播放列表（PlayList.mxml）文件的内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TileList xmlns:mx="http://www.adobe.com/2006/mxml" itemRenderer="myComp-
onents.PlayListBox" direction="vertical" selectedIndex="0" columnCount="1"
rowCount="3" height="380">
</mx:TileList>
```

播放列表的单项文件基于 Hbox 组件，该文件用来显示信息和接受鼠标单击事件。播放列表绑定了数据源的 pic、name 及 desciption 属性，当鼠标单击该项时，发送以该项的数据为参数的 PlayListEvent 事件。该文件的代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="338" height="105"
borderStyle="solid" borderThickness="2" borderColor="#FBB100" click="Change-
ToPlayMe(event)" buttonMode="true" useHandCursor="true">
    <mx:Script>
        <![CDATA[
            import myEvents.PlayListEvent;
            private function ChangeToPlayMe(event:MouseEvent):void{
                this.parentApplication.dispatchEvent(new PlayListEvent(eve-
                nt.currentTarget.data));
            }
        ]]>
    </mx:Script>
    <mx:Image height="101" width="160" source="{data.@pic}"/>
    <mx:VBox width="100%" height="100%">
        <mx:Text text="{data.@name}"/>
        <mx:Text text="{data.@desciption}" width="100%" textAlign="left"
```

```
            height="100%" fontSize="12"/>
        </mx:VBox>
    </mx:HBox>
```

16.4.3 调节器

调节器为一个基于 TitleWindow 的自定义组件，包括 4 个 Hslider 组件，用来调节亮度、对比度等，2 个 Button 组件，用来确认和还原调节数据。该调节器的布局和设置代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="278" height="200" showCloseButton="true" title="调节器" cornerRadius=
"2" close="CloseSetting()" borderColor="#FFFFFF" fontSize="15" backgroundCo-
lor="#000000" buttonMode="true" useHandCursor="true">
    <mx:Style source="css/Setting.css">
    </mx:Style>
    <mx:HSlider x="75" y="97" id="sldBright" minimum="-100" maximum="100"
    snapInterval="1" value="0" change="setBright()"styleName="settingHSlide-
    r" showDataTip="false" buttonMode="true" useHandCursor="true"/>
    <mx:Label x="27" y="95" text="亮  度: " fontSize="12"/>
    <mx:HSlider x="75" y="71" id="sldContrast" minimum="-100" maximum="100"
    snapInterval="1" value="0" change="setBright()" styleName="settingHSlid-
    er" showDataTip="false" buttonMode="true" useHandCursor="true"/>
    <mx:HSlider x="75" y="44" id="sldSaturation" minimum="-100" maximum="100"
    snapInterval="1" value="0" change="setBright()" styleName="settingHSlid-
    er" showDataTip="false" buttonMode="true" useHandCursor="true"/>
    <mx:Label x="26" y="44" text="饱和度: " fontSize="12"/>
    <mx:HSlider x="76" y="18" id="sldHue" minimum="-100" maximum="100" snap-
    Interval="1" value="0" change="setBright()" showDataTip="false" useHandC-
    ursor="true" styleName="settingHSlider" buttonMode="true"/>
    <mx:Label x="27" y="18" text="反转度: " fontSize="12"/>
    <mx:Label x="26" y="71" text="对比度: " fontSize="12"/>
    <mx:Button x="146" y="137" label="还原" fontSize="12" click="reset()" but-
    tonMode="true" mouseChildren="false" useHandCursor="true"/>
    <mx:Button x="66" y="137" label="确定" fontSize="12" click="CloseSetting()"
    buttonMode="true" mouseChildren="false" useHandCursor="true"/>
</mx:TitleWindow>
```

上述代码中，为 4 个 Hslider 组件的 change 事件指定了处理函数。当用户调节滑块位置时，调用 setBright()函数。另外，还有该调节器的【关闭】按钮单击事件，【确定】和【还原】按钮单击事件，都指定了处理函数。下面，添加 Script 脚本，编写处理函数。首先来看 CloseSetting()函数，该函数实现移除场景中创建的调节器对象。代码如下所示。

```
<mx:Script>
```

```
        <![CDATA[
            import mx.events.SliderEvent;
            import myEvents.SettingEvent;
            import mx.managers.PopUpManager;
            private function CloseSetting():void
            {
                if(this.isPopUp){
                    PopUpManager.removePopUp(this);
                }
            }
        ]]>
    </mx:Script>
```

然后添加 setBright()函数。该函数创建 SettingEvent 事件对象，该对象的 4 个参数分别为 4 个 Hslider 组件的值。然后发送该事件对象到事件流，由调节器监听。代码如下所示。

```
    private function setBright():void
    {
        dispatchEvent(new SettingEvent(this.sldBright.value,this.sldContrast.
        value,this.sldSaturation.value,this.sldHue.value));
    }
```

添加 reset()函数。该函数还原 4 个 Hslider 组件的值，然后调用 setBright()函数，请求还原视频的色彩。代码如下所示。

```
    private function reset():void
    {
        sldBright.value=0;
        sldContrast.value=0;
        sldSaturation.value=0;
        sldHue.value=0;
        setBright();
    }
```

16.5　主程序设计

经过上面 4 节的准备，播放器功能模块已经创建完成，现在就要将其添加到主程序中。主程序中包括了一个自定义的播放和控制器、一个播放列表和控制播放列表显示的按钮。其布局和设置代码如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
    xmlns:ns1="myComponents.*" creationComplete="init()">
        <ns1:Player id="VideoPlayer" x="10" y="10" useHandCursor="true">
```

```
        </ns1:Player>
    <mx:Button x="341" y="12" id="btShowPlaylist" label="查看更多" fontSize=
    "12" click="ChangePlayList(false)" useHandCursor="true" buttonMode="tr-
    ue" mouseChildren="false"/>
    <ns1:PlayList x="420" y="10" dataProvider="{XMLPlayList.video}" selected-
    Index="0" id="myPlayList" visible="false"/>
</mx:Application>
```

添加 Script 脚本，提供数据和处理各种组件的请求。首先来添加变量，包括变量 mysetting，是调节器对象，用来在需要时以弹出形式添加到场景中；变量 XMLPlayList，是播放列表的数据源；变量 nowPlaying 是正在播放的视频名称；变量 fullscreenTimer 是一个 6000 毫秒的计数器，用来在全屏状态下记录鼠标空闲的时间。代码如下所示。

```
<mx:Script>
    <![CDATA[
        import mx.events.ResizeEvent;
        import myClass.ColorMatrix;
        import myEvents.SettingEvent;
        import myEvents.PlayerEvent;
        import myEvents.PlayListEvent;
        import myComponents.Setting;
        import mx.managers.PopUpManager;
        import flash.display.StageDisplayState;
        private var mysetting:Setting=new Setting;
        [Bindable]
        private var XMLPlayList:XML;
        [Bindable]
        private var nowPlaying:String;
        private var fullscreenTimer:Timer=new Timer(6000,1);
    ]]>
</mx:Script>
```

添加场景创建完成时的初始化函数 init()。该函数加载外部 XML 文件，用来读取播放列表数据源，为场景添加不同参数的事件监听器，交由不同的函数进行处理。代码如下所示。

```
private function init():void{
    LoadMyXML("myXML/playlist.xml");
    this.addEventListener("ChangePlayList",ChangeToPlay);
    this.addEventListener("Play",Play);
    this.addEventListener("Pause",Pause);
    this.addEventListener("PlayComplete",PlayNext);
    this.addEventListener("RePlay",RePlay);
    this.addEventListener("ShowSetting",showSetting);
    this.addEventListener("FullScreen",FullScreen);
    this.addEventListener("NormalScreen",NormalScreen);
    this.VideoPlayer.addEventListener(MouseEvent.MOUSE_OVER,SetHandCursor);
```

```
        this.addEventListener(ResizeEvent.RESIZE,ChangeState);
        this.fullscreenTimer.addEventListener(TimerEvent.TIMER_COMPLETE,FullSc-
        reenTimerEnd);
    }
```


添加 LoadMyXML()函数，该函数创建一个 URLLoader 对象，为该对象添加是否加载完成监听器，当加载完成时执行 HandleLoadComplete()函数。代码如下所示。

```
    private function LoadMyXML(xmlpath:String):void{
        var loader:URLLoader=new URLLoader();
        loader.dataFormat=URLLoaderDataFormat.TEXT;
        loader.addEventListener(Event.COMPLETE,HandleLoadComplete);
        loader.load(new URLRequest(xmlpath));
    }
```

HandleLoadComplete()函数在加载 XML 文件完成时执行。首先将加载的数据储存到 XMLPlayList 变量。这样，播放列表的信息得到更新。然后调用 PlayThis()函数，传递 XML 文件的第一条信息。代码如下所示。

```
    private function HandleLoadComplete(event:Event):void{
        XMLPlayList=new XML(event.target.data);
        PlayThis(XMLPlayList.video[0].@url.toString(),XMLPlayList.video[0].@na
        me.toString());
    }
```

PlayThis()函数将得到的 path 参数作为播放器的视频地址；name 参数赋予播放器的 nowPlaying 成员，也就是其 Panel 的 Title 属性。然后调用 InitPlayer()函数，初始化播放器的状态为播放状态。代码如下所示。

```
    private function PlayThis(path:String,name:String):void{
        this.VideoPlayer.nowPlaying=name;
        this.VideoPlayer.sliderVideoPosition.value=0;
        this.VideoPlayer.myVideoDisplay.source=path;
        this.VideoPlayer.myVideoDisplay.playheadTime=0;
        InitPlayer(true);
    }
```

InitPlayer()函数根据传递的参数改变播放器的相关组件状态。例如当传递的参数为真时，设置播放器的 VideoDisplay 组件不透明，显示暂停按钮、隐藏播放和重播按钮，为播放器的 isvideoplayed 变量赋 true 值。代码如下所示。

```
    private function InitPlayer(isplay:Boolean):void{
        if(isplay){
            this.VideoPlayer.myVideoDisplay.alpha=1;
            this.VideoPlayer.canvasPause.visible=true;
            this.VideoPlayer.canvasPlay.visible=false;
            this.VideoPlayer.canvasRePlay.visible=false;
```

```
            this.VideoPlayer.isvideoplayed=true;
            this.VideoPlayer.btSmallPlay.styleName="smallpause";
        }else{
            this.VideoPlayer.canvasPause.visible=false;
            this.VideoPlayer.canvasPlay.visible=true;
            this.VideoPlayer.canvasRePlay.visible=false;
            this.VideoPlayer.isvideoplayed=false;
            this.VideoPlayer.btSmallPlay.styleName="smallplay";
        }
    }
```

添加 ChangeToPlay()函数。在单击播放列表某一项时，由自定义的单项组件发送参数为 ChangePlayList 的 PlayListEvent 事件。主程序监听到该事件后，交由 ChangeToPlay()函数处理。该函数调用 PlayThis()函数，播放选中播放列表的视频。代码如下所示。

```
    private function ChangeToPlay(ctpevent:PlayListEvent):void{
        PlayThis(ctpevent.video.@url.toString(),ctpevent.video.@name.toString(
        ));
    }
```

添加 Play()和 Pause()函数。这两个函数简单，改变播放器的播放状态。然后调用 InitPlayer()函数，改变相关组件状态。播放和暂停时的效果如图 16-3 和图 16-4 所示，代码如下所示。



图 16-3　播放状态下的效果

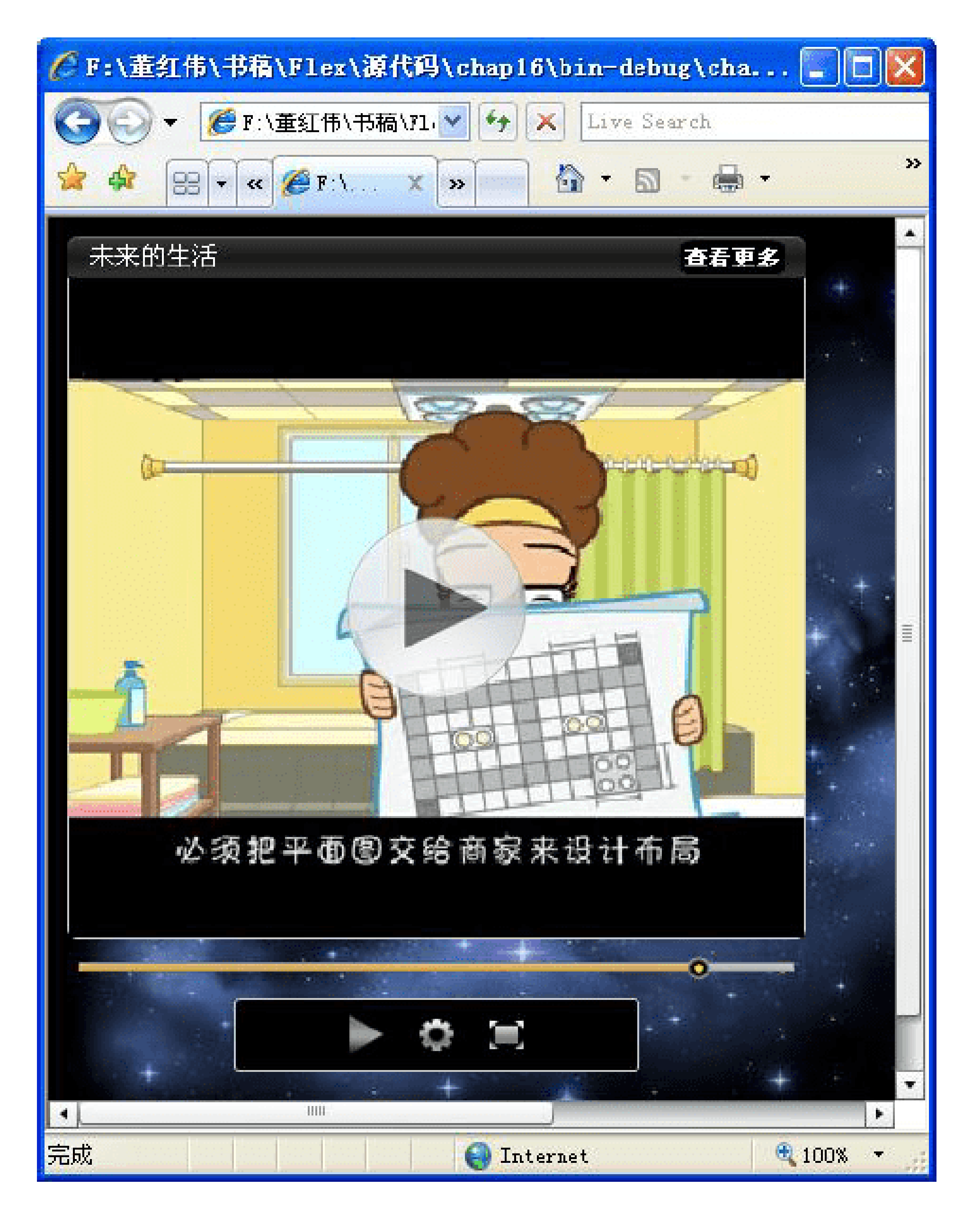


图 16-4　暂停状态下的效果

```
    private function Play(playevent:PlayerEvent):void{
        this.VideoPlayer.myVideoDisplay.play();
        InitPlayer(true);
```

```
    }
    private function Pause(playevent:PlayerEvent):void{
        this.VideoPlayer.myVideoDisplay.pause();
        InitPlayer(false);
    }
```


添加 PlayNext()和 RePlay()函数。PlayNext()函数是在当前视频播放完成后执行，显示【重播】按钮，并调用 ChangePlayList()函数，强制显示播放列表。当前视频播放完成后的效果如图 16-5 所示。RePlay()函数是在鼠标单击【重播】按钮时执行，完成重播的功能。两个函数的定义代码如下所示。



图 16-5　当前视频播放完成后的效果

```
    private function PlayNext(playevent:PlayerEvent):void{
        this.VideoPlayer.canvasRePlay.visible=true;
        ChangePlayList(true);
        this.VideoPlayer.myVideoDisplay.alpha=0.4;
    }
    private function RePlay(playevent:PlayerEvent):void{
        this.VideoPlayer.canvasRePlay.visible=false;
        this.VideoPlayer.myVideoDisplay.play();
        InitPlayer(true);
    }
```

ChangePlayList()函数用来处理对播放列表的显示和隐藏。这需要根据场景的当前状态，例如，当场景处于全屏模式下时，发送退出全屏请求；改变 btShowPlaylist 对象的 label 属性为“隐藏更多”；显示播放列表对象。代码如下所示。

```
    private function ChangePlayList(ismustshow:Boolean):void{
        if(btShowPlaylist.label=="查看更多"){
```

```
            if(stage.displayState==StageDisplayState.FULL_SCREEN){
                this.dispatchEvent(new PlayerEvent("NormalScreen"));
            }
            btShowPlaylist.label="隐藏更多";
            this.myPlayList.visible=true;
        }else{
            if(!ismustshow){
                btShowPlaylist.label="查看更多";
                this.myPlayList.visible=false;
            }
        }
    }
```

添加 showSetting()函数，该函数是在单击【设置】按钮时执行的函数。该函数首先调用 PopUpManager 对象的 removePopUp()方法，将场景中存在的 mysetting 对象移除掉，然后创建新的弹出窗口。代码如下所示。

```
    private function showSetting(playerevent:PlayerEvent):void{
        PopUpManager.removePopUp(mysetting);
        mysetting.x=this.width/2-mysetting.width/2;
        mysetting.y=this.VideoPlayer.myVideoDisplay.height/2-mysetting.height/
        2;
        mysetting.addEventListener("ChangeSetting",ChangeSetting);
        PopUpManager.addPopUp(mysetting,this,false);
    }
```

上述代码中，为播放控制器组件添加了事件监听器，监听参数为 ChangeSetting 的事件，交由函数 ChangeSetting()处理。该函数创建 ColorMatrix 对象，通过调用该对象的 adjustColor()方法，改变该对象的值。然后将该对象添加到播放器的滤镜，实现调节视频色彩的功能。效果如图 16-6 所示，代码如下所示。

```
    private function ChangeSetting(settingevent:SettingEvent):void{
        var cm:ColorMatrix=new ColorMatrix();
        cm.adjustColor(settingevent.Bright,settingevent.Contrast,settingevent.
        Saturation,settingevent.Hue);
        this.VideoPlayer.myVideoDisplay.filters=[new ColorMatrixFilter(cm)];
    }
```

添加 FullScreen()函数，该函数是在主程序监听到参数为 FullScreen 的事件时执行。该函数通过为场景 stage 的 displayState 属性，赋予 StageDisplayState.FULL_SCREEN 值，实现场景的全屏效果；改变主程序的当前 State 值为 FullScreen，此 State 将在下文中创建；改变播放器的当前 State 值为 FullScreen；启动 fullscreenTimer 对象的计数；为主程序添加鼠标移动的事件监听器，监听到时，交由 StopTimer()函数执行。全屏后的效果如图 16-7 所示，FullScreen()函数的定义代码如下所示。

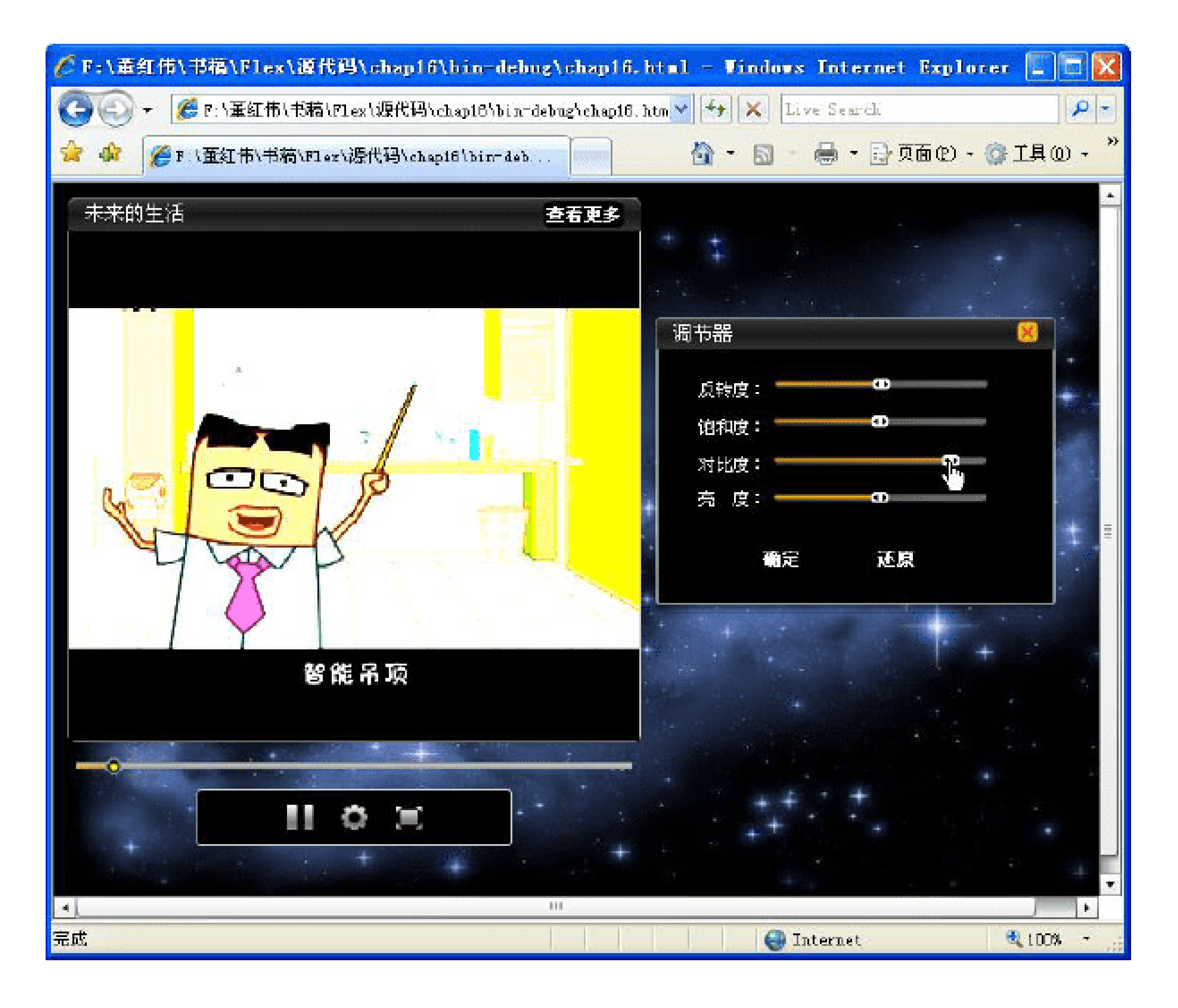


图 16-6　调节视频的对比度效果

```
private function FullScreen(playerevent:PlayerEvent):void{
    stage.displayState=StageDisplayState.FULL_SCREEN;
    this.currentState="FullScreen";
    this.VideoPlayer.currentState="FullScreen";
    this.fullscreenTimer.start();
    this.addEventListener(MouseEvent.MOUSE_MOVE,StopTimer);
}
```

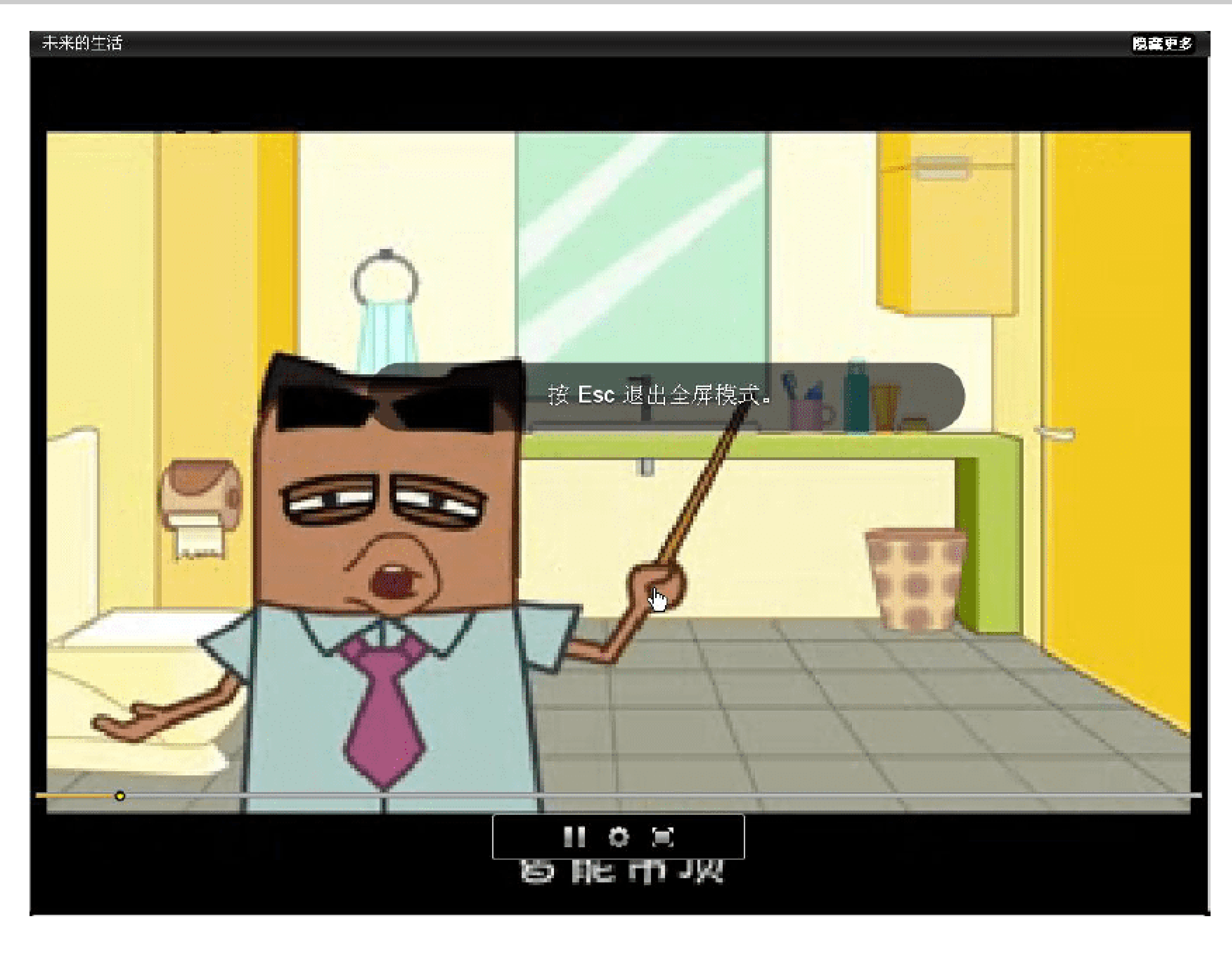
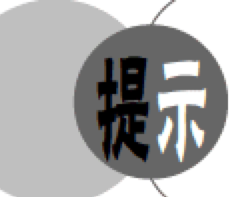


图 16-7　全屏后的效果



在 IE 浏览器中运行程序时，实现全屏显示必须改变 HTML 文件的代码。将页面中所有 allowScriptAccess 参数删除，添加值为 true 的 allowFullScreen 参数。

添加 NormalScreen()函数，该函数与 FullScreen()函数相反，实现场景由全屏到非全屏的转变。代码如下所示。

```
private function NormalScreen(playerevent:PlayerEvent):void{
    stage.displayState=StageDisplayState.NORMAL;
    this.fullscreenTimer.stop();
    this.removeEventListener(MouseEvent.MOUSE_MOVE,StopTimer);
    this.VideoPlayer.hbox.alpha=1;
    this.VideoPlayer.progressbar1.alpha=1;
    this.VideoPlayer.sliderVideoPosition.alpha=1;
    this.currentState=null;
    this.VideoPlayer.currentState=null;
}
```

ChangeState()函数是在场景被重绘大小时执行。由于在全屏模式下，用户可以按键盘上的 Esc 键退出全屏，或通过其他按键退出全屏，例如 Alt+Tab 组合键。为每一个可以退出全屏的按键添加事件监听器，有些繁琐。如果不添加任何监听器，当用户通过这些按键退出全屏时，场景仍处于名为 FullScreen 的 State 状态，这样是用户不愿意看到的。ChangeState()函数根据 stage 的 displayState 属性的当前值，如果为非全屏，并且当前场景处于名为 FullScreen 的 State 状态，那么将场景转到非全屏状态。代码如下所示。

```
private function ChangeState(resizeevent:ResizeEvent):void{
    if(stage.displayState==StageDisplayState.NORMAL&&this.currentState=="F-
    ullScreen"){
        this.fullscreenTimer.stop();
        this.removeEventListener(MouseEvent.MOUSE_MOVE,StopTimer);
        this.VideoPlayer.hbox.alpha=1;
        this.VideoPlayer.progressbar1.alpha=1;
        this.VideoPlayer.sliderVideoPosition.alpha=1;
        this.currentState=null;
        this.VideoPlayer.currentState=null;
    }
}
```

添加 FullScreenTimerEnd()函数，该函数在 fullscreenTimer 对象设置的时间到期时执行。主要是隐藏播放器的控制器、播放进度条、加载进度条，效果如图 16-8 所示，代码如下所示。

```
private function FullScreenTimerEnd(timerevent:TimerEvent):void{
    this.VideoPlayer.hbox.alpha=0;
    this.VideoPlayer.progressbar1.alpha=0;
    this.VideoPlayer.sliderVideoPosition.alpha=0;
}
```



图 16-8　fullscreenTimer 对象设置的时间到期时效果

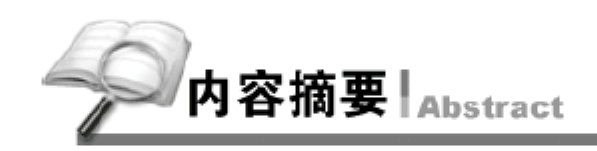
添加 StopTimer()函数，该函数在鼠标移动时执行，并且在程序中只有当场景在全屏模式下才执行。该函数将 fullscreenTimer 对象复位，显示控制器、播放进度条和加载进度条。然后重新开启 fullscreenTimer 对象的计数。代码如下所示。

```
private function StopTimer(moveevent:MouseEvent):void{
    this.fullscreenTimer.reset();
    this.VideoPlayer.hbox.alpha=1;
    this.VideoPlayer.progressbar1.alpha=1;
    this.VideoPlayer.sliderVideoPosition.alpha=1;
    this.fullscreenTimer.start();
}
```

为主程序添加名为 FullScreen 的 State，在该 State 下，视频播放器宽度和高度都为 100%，代码如下所示。

```
<mx:states>
    <mx:State name="FullScreen">
        <mx:RemoveChild target="{myPlayList}"/>
        <mx:SetProperty target="{VideoPlayer}" name="x"/>
        <mx:SetProperty target="{VideoPlayer}" name="y"/>
        <mx:SetProperty target="{VideoPlayer}" name="width" value="100%"/>
        <mx:SetProperty target="{VideoPlayer}" name="height" value="100%"/>
        <mx:SetProperty target="{btShowPlaylist}" name="x"/>
        <mx:SetStyle target="{btShowPlaylist}" name="right" value="10"/>
        <mx:SetProperty target="{btShowPlaylist}" name="y" value="2"/>
    </mx:State>
</mx:states>
```

第17章 视频展示网站



随着网络带宽的增加，在线视频播放变得越来越受到网友的欢迎，特别是 FLV 格式视频的出现，将网络视频推上了一个新的层次。视频展示网站从此层出不穷，如土豆网、优酷网等。本章节将讲述如何通过 Flex 制作视频展示网站。



- 了解视频展示网站的需求分析
- 了解系统的结构设计
- 掌握系统数据库表的设计
- 熟悉数据库类的设计
- 掌握如何使用 ASP.NET 开发服务器端
- 掌握网站中的事件处理类设计
- 掌握用户模块的设计
- 掌握实现可拖动代码
- 了解分页组件
- 掌握 Flex 中如何与服务器端通信
- 掌握后台添加视频的实现方法

17.1 系统概述

上一章介绍的视频播放器，没有与服务器端交互，只能称为单机版播放器。本章介绍的视频展示网站的服务器端使用 ASP.NET 3.5，数据库使用 SQL Server 2005。网站分为前台和后台两个页面。前台用来显示视频，供浏览者查看，后台可以对视频和分类进行编辑。

17.1.1 需求分析

作为时下较流行的视频展示网站，和一般的相册展示网站功能类似，只不过照片换成了视频。视频展示网站包括大量的视频，对视频归类，浏览者按照网站的指示图或者个人搜索，

可以迅速找到所需的视频。注册用户可以将感兴趣的视频收藏，很方便地找到此视频，不需要每次都搜索。

根据以上功能需求，可以列出前台需要的模块，主要包括以下几部分。

- 用户注册模块
- 用户登录模块
- 视频列表模块
- 视频播放模块

用户不需要登录就可以浏览视频，只有登录后，才可以收藏视频和修改个人信息。按照需要实现的功能，可以设计出首页的界面，包括以下内容。

- 用户注册和登录按钮
- 视频分类菜单
- 视频搜索栏
- 视频列表显示区域
- 视频播放显示区域

Flex 基于客户端开发，不能应用服务器端的 Session 对象，这样记录用户登录就没有服务器端直接开发方便。为了实现这一功能，本网站在服务器端仍然采用 Session 方式，客户端采用应用程序中的公有变量。

后台结构功能比较简单，主要包括管理员登录、视频和类别添加、视频和类别管理，还应包括用户管理。管理员登录和用户管理是不能少的，由于前台中已经有过用户登录模块，用户管理和视频管理类似，在介绍后台时，不会再重复。后台所需要的模块包括以下几部分。

- 管理员登录模块
- 添加视频模块
- 添加类别模块
- 修改和删除视频模块
- 修改和删除类别模块
- 管理用户模块

后台界面也比较简单，开始为管理员登录界面，登录成功显示用户、视频及类别管理按钮等。

17.1.2　结构设计

出于安全因素，网站的前台和后台采用了两个不同页面，这样将普通用户和管理员的职能分开。前台主要实现在线视频播放，视频播放可以使用第 16 章中的 FLV 播放器。当然，播放之前，用户需要通过搜索或收藏夹，找到需要播放的视频。搜索视频可分为类别搜索和关键字搜索。为了修改方便和实现代码重用，作者推荐编写程序时，能够写成自定义组件的模块，一定要编写成自定义组件。具有共性的组件或类存放到单独的文件夹中，方便查找。

本网站是在 Flex Builder 3 中，通过创建 Flex 项目构建的。除了软件自动生成的文件夹外，

系统需要的文件夹列表及其描述如表 17-1 所示。

表 17-1　系统文件夹列表和描述

文件夹	功能
App_Code	存放系统服务器端处理数据库类文件
App_Data	存放系统数据库文件
src/css	存放系统 CSS 样式表文件
src/flv	存放上传的视频及缩略图文件
src/images	存放系统用到的图片文件
src/myComponents	存放自定义组件文件
src/myEvents	存放自定义事件类文件
src/myTheme	存放自定义主题文件
src/myXML	存放用户管理菜单文件
src/Server	存放服务器端程序文件

前台页面的功能，主要包括用户登录和注册、视频分类、视频搜索、视频播放等。后台的功能包括对视频和类别的管理，详细内容如图 17-1 所示。

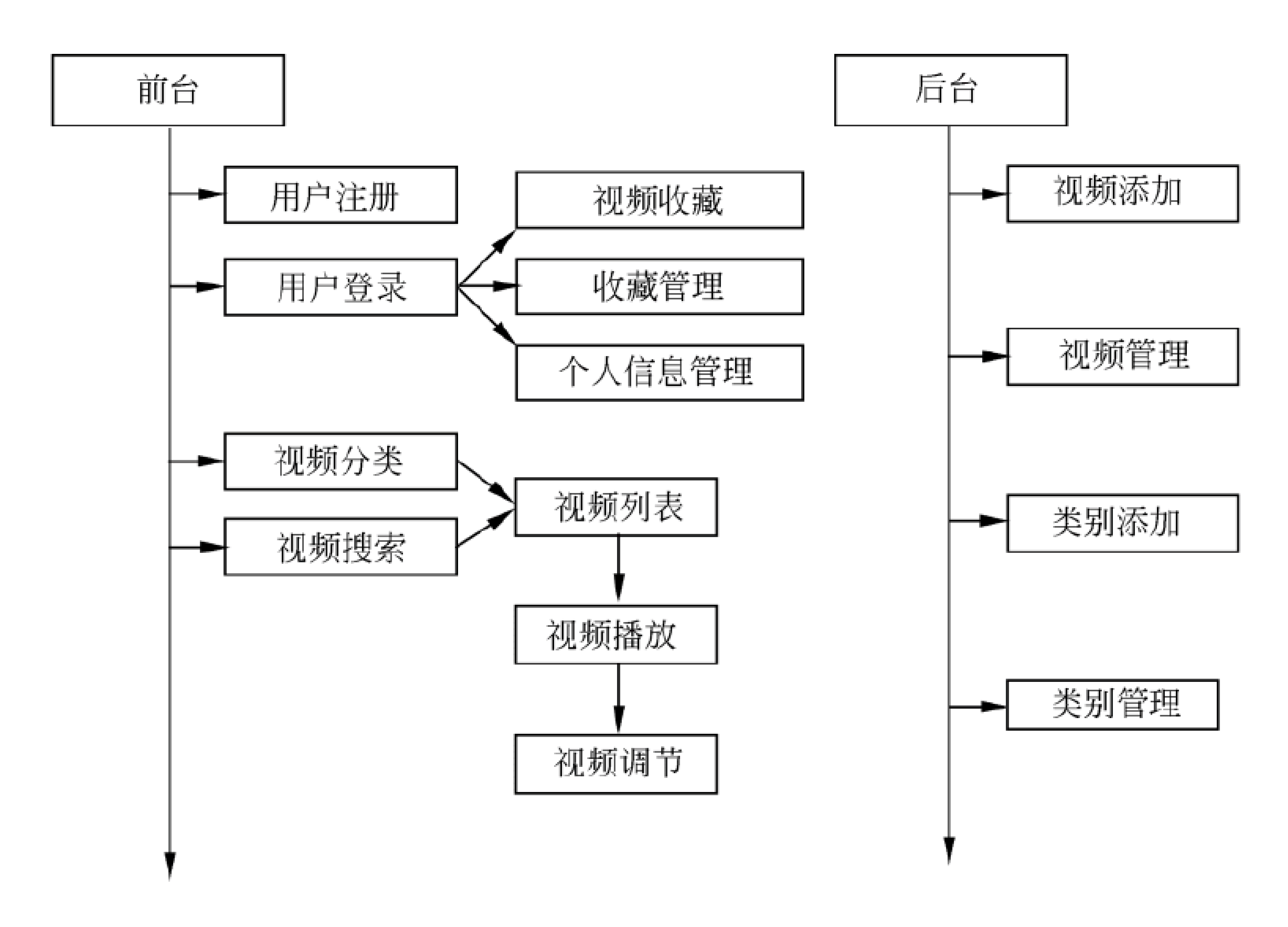


图 17-1　系统结构功能图

17.2　数据库和数据库类设计

本网站采用三层结构的模式进行开发，即用户界面层（Flex 端）、业务逻辑层（ASP.NET 端）和数据存储层（SQL Server 2005 端）。其中，Flex 端最为复杂，涉及到的知识面较广，下面从简到难，一步一步实现这三层。首先设计数据库。

17.2.1　数据库设计

本网站中，需要储存到数据库中的数据包括用户资料、视频类别资料、视频资料和用户收藏信息。需要创建的表为用户、视频类别、视频和用户收藏 4 张表，下面在 SQL Server 2005 中分别进行创建。

□ **Users 表**

Users 表保存的是用户的编号、用户名、密码、E-mail 及联系电话等，如表 17-2 所示。

表 17-2　Users 表

字段	数据类型	长度	允许空	备注
id	int	4	否	用户编号，自增长
username	varchar	20	否	用户名
password	varchar	100	否	用户的密码
email	varchar	100	是	用户的 E-mail
phone	varchar	20	是	用户的电话

□ **ClassType 表**

ClassType 表保存的是视频类别信息，如表 17-3 所示。

表 17-3　ClassType 表

字段	数据类型	长度	允许空	备注
id	int	4	否	类别编号，自增长
classname	varchar	50	否	类别名
parentid	int	4	否	类别的上级分类编号

□ **Videos 表**

Videos 表保存的是视频序号、所属类别、名称、地址和缩略图地址等，如表 17-4 所示。

表 17-4　Videos 表

字段	数据类型	长度	允许空	备注
id	int	4	否	视频编号，自增长
classid	int	4	否	视频类别编号
videoname	varchar	50	是	视频名
videosrc	varchar	50	是	视频地址
picsrc	varchar	50	是	视频缩略图地址

□ **Favorite 表**

Favorite 表保存的是收藏编号、用户编号、视频编号和添加时间等，如表 17-5 所示。

表 17-5　Favorite 表

字段	数据类型	长度	允许空	备注
id	int	4	否	收藏编号，自增长
userid	int	4	否	用户编号
videoid	int	4	否	视频编号
adddatetime	varchar	255	是	添加时间

上述 4 个表已经可以实现整个网站的需求，但是为了查询更方便，需要创建 3 个视图，如下所示。

- **view_classtype 视图**

view_classtype 视图将 ClassType 表与自身内联，得到二级类别的列表，列表中包括上级分类的信息。其 SQL 代码如下所示。

```
SELECT a.id, a.classname, a.parentid, b.classname AS parentclassname
FROM ClassType AS a INNER JOIN ClassType AS b ON a.parentid = b.id
```

- **View_videos_classtype 视图**

View_videos_classtype 视图将 view_classtype 视图与 Videos 表内联，得到视频列表，列表中包括视频别的全部信息。其 SQL 代码如下所示。

```
SELECT view_classtype.classname, view_classtype.parentclassname, Videos.id,
Videos.classid,
Videos.videoname, Videos.videosrc, Videos.picsrc, view_classtype.parentid
FROM Videos INNER JOIN view_classtype ON Videos.classid=view_classtype.id
```

- **view_favorite_video 视图**

view_favorite_video 视图将 Favorite 表与 Videos 表内联，得到用户收藏列表，列表中包括收藏视频的信息。其 SQL 代码如下所示。

```
SELECT Favorite.userid, Favorite.videoid, Favorite.adddatetime, Videos.cla-
ssid, Videos.videoname,
Videos.videosrc, Videos.picsrc FROM Favorite INNER JOIN Videos ON Favorite.
videoid=Videos.id
```

17.2.2 数据库类设计

在使用面向对象语言编程的过程中，通过很多方法可以实现代码的封装和复用，类是使用最多的。ASP.NET 将任何的 Web 窗体页都以类的形式来组织，因此可以将一些常用的方法封装到一个类中，从而提高开发的效率和性能。

数据库类就是其中最具代表性的一个，因为系统中大部分的业务逻辑最终处理的结果都需要保存到数据库中，像添加视频类别、视频资料及用户资料等。还有更新类别、视频资料、删除资料等，这些业务逻辑都比较简单，这里直接在 Web 窗体的后台代码中实现。

数据库类实现的是对数据库的基础操作，例如与数据库的连接、执行 SQL 语句并得到各种返回值以利于进一步使用数据库返回的结果。创建一个名为 SQLHelper 的类，将其存放在网站根目录的 App_Code 子目录中，然后添加所需的命名空间，如下所示。

```
using System;
using System.Collections;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
```

接下来创建这个 SQLHelper 类，这里为类添加一个命名空间 SQLServerDAL，这样可轻松管理和组织多个数据库类，如下所示。

```
namespace SQLServerDAL
{
   public abstract class SQLHelper
  {
   // 在这里编写类的代码实现数据库操作
   }
}
```

上代码所示仅为一个空类，其中 abstract 关键字指定这是一个抽象类。使用 abstract 还可以指定抽象方法，但抽象方法只能存在于抽象类中。使用 abstract 关键字创建的抽象类不能实例化，它的作用是提供多个派生类来共享基类的公共定义，要使用抽象类必须继承这个类。

在 SQLHelper 类中添加数据成员，这里仅包含有一个成员 connectionString，它存储的是实例数据库的连接字符串。该连接字符串为 web.config 文件中定义的名为 VideoConnectionString 的连接字符串，如下所示。

```
protected static string connectionString = ConfigurationManager.ConnectionStr-
ings["VideoConnectionString"].Connection   String;
```

web.config 文件中定义名为 VideoConnectionString 的连接字符串，代码如下所示。

```
<connectionStrings>
    <add name="VideoConnectionString" connectionString="Data Source=.\SQLEX-
    PRESS;AttachDbFilename=|DataDirectory|\VideoDataBase.mdf;Integrated Se-
    curity=True;User Instance=True" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

protected 修饰符是一个保护成员访问修饰符。只有声明 protected 类中及声明 protected 类型成员类的派生类中才可以访问该成员。也就是说，protected 类型的类成员只有在本类和派生类中才能访问，外部代码无法访问。

向类中添加一个用于执行 SQL 语句的方法 ExecuteSql()，它带有一个参数来指定 SQL 语句，返回执行后影响的记录数。

```
/**//// <summary>
/// 执行 SQL 语句，返回影响的记录数
/// </summary>
/// <param name="SQLString">SQL 语句</param>
/// <returns>影响的记录数</returns>
public static int ExecuteSql(string SQLString)
 {
   using (SqlConnection connection = new SqlConnection(connectionString))
    {
      using (SqlCommand cmd = new SqlCommand(SQLString, connection))
       {
```

```
            try
             {
                connection.Open();
                int rows = cmd.ExecuteNonQuery();
                return rows;
             }
            //处理错误情况
            catch (System.Data.SqlClient.SqlException E)
             {
                //关闭数据库连接，显示错误信息
                connection.Close();
                throw new Exception(E.Message);
             }
        }
    }
}
```

ExecuteSql()方法使用<summary>形式来组织注释，这样符合.NET 的定义规范，可以方便地生成 XML 文档，从而提高协作性。也即是如果将类编译成 DLL 文件，再次使用该类时则看不到源码，可通过 XML 文档来查看其中包含的各种方法、方法作用以及参数的含义等。

从上述 ExecuteSql()方法的代码中可以看出，该方法适合于执行数据库修改时的情况，像 Insert 语句、Update 语句和 Delete 语句等。

如果要执行 Select 语句则可以使用下面介绍的 ExecuteReader()方法。该方法用于执行返回结果有若干条时，其代码如下所示。

```
/**//// <summary>
/// 执行查询语句，返回 SqlDataReader
/// </summary>
/// <param name="strSQL">查询语句</param>
/// <returns>SqlDataReader</returns>
public static SqlDataReader ExecuteReader(string strSQL)
 {
   SqlConnection connection = new SqlConnection(connectionString);
   SqlCommand cmd = new SqlCommand(strSQL, connection);
   try
    {
      connection.Open();
      SqlDataReader myReader = cmd.ExecuteReader();
      return myReader;
    }
   catch (System.Data.SqlClient.SqlException e)
    {
      throw new Exception(e.Message);
    }
}
```

如果要执行一条计算查询结果语句，并返回一个查询结果，则使用 GetSingle()函数，该函数的代码如下所示。

```
/**//// <summary>
/// 执行一条计算查询结果语句，返回查询结果（object）。
/// </summary>
/// <param name="SQLString">计算查询结果语句</param>
/// <returns>查询结果（object）</returns>
public static object GetSingle(string SQLString)
 {
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        using (SqlCommand cmd = new SqlCommand(SQLString, connection))
        {
            try
            {
               connection.Open();
               object obj = cmd.ExecuteScalar();
               if ((Object.Equals(obj, null)) || (Object.Equals(obj,System.DBN-
               ull.Value)))
                {
                   return null;
                }
                else
                {
                    return obj;
                }
            }
            catch (System.Data.SqlClient.SqlException e)
            {
                connection.Close();
                throw new Exception(e.Message);
            }
        }
    }
}
```

因为在数据库中，与用户相关的表，存储的都是用户编号，而页面中记录的用户信息为用户名。所以，本实例中经常需要根据用户名查询用户编号，将这个功能添加到数据库类中，方便调用，其代码如下所示。

```
Public  static string GetUserId(string username) {
        string sql = "select id from [Users] where username='"+username+"'";
        string userid = Convert.ToString(GetSingle(sql));
        return userid;
}
```

17.3 服务器端程序设计

本实例中的数据库非常简单，服务器端程序也不是那么复杂，主要分为用户、视频分类、视频列表和收藏等处理。每个处理单独创建文件，方便程序员查找，也可以提高程序的运行速度。下面讲解其中的几个程序文件。

17.3.1 处理用户程序文件

本网站采用常用的 Session（会话）方式，记录用户登录信息。服务器端需要的用户相关功能，包括注册、登录、判断是否登录、用户信息获得和修改、用户注销等。

用户注册程序文件为本系统中的 Reg.aspx，主要是获得用户提交的信息，将这些信息保存到数据库中。该程序文件的代码如下所示。

```
protected void Page_Load(object sender, EventArgs e)
    {
        string username = Convert.ToString(Request.QueryString["usernam-
        e"]);
        string password = Convert.ToString(Request.QueryString["password
        "]);
        string email = Convert.ToString(Request.QueryString["email"]);
        string sql = "select * from [Users] where username='" + username+"'" ;
        SqlDataReader reader = SQLHelper.ExecuteReader(sql);
        //用户名已被注册情况
        if (reader.Read())
        {
            Response.Write("<result><state>usernameRegisted</state></res-
            ult>");
            reader.Close();
            reader.Dispose();
        }
        //保存用户信息到数据库
        else
        {
            sql = "insert into [Users] (username,password,email) values('"
                +username+"','"+password+"','"+email+"')";
            SQLHelper.ExecuteSql(sql);
            Response.Write("<result><state>registok</state></result>");
            Session["username"] = username;
        }
    }
```

用户成功注册后，自动记录用户名到 Session 中。前台仍然需要用户登录功能，服务器端处理用户登录的程序文件为 CheckLogin.aspx，该文件的代码如下所示。

```
protected void Page_Load(object sender, EventArgs e)
        {
            string username = Convert.ToString(Request.QueryString["usernam-
            e"]);
            string password = Convert.ToString(Request.QueryString["password
            "]);
string sql = "select * from [Users] where username='" + username + "' and password='"
+ password + "'";
            SqlDataReader reader = SQLHelper.ExecuteReader(sql);
            //成功登录
            if (reader.Read())
            {
Response.Write("<result><state>yes</state><username>"+username+"</username
></result>");
                reader.Close();
                reader.Dispose();
                Session["username"] = username;
            }
           //登录失败
            else
            {
                Response.Write("<result><state>no</state></result>");
            }
        }
```

判断用户是否登录的处理文件为 CheckSession.aspx。用户注销的处理文件为 LoginOut.aspx，该文件将 Session 清空。这两个文件比较简单，这里不再显示代码。用户信息获得和修改的处理文件为 GetUserInfo.aspx，其代码如下所示。

```
string returnstr;
protected void Page_Load(object sender, EventArgs e)
{
   if(Convert.ToString(Session["username"])==null||
   Convert.ToString(Session["username"])==""){
       return;
   }
    string action = Convert.ToString(Request.QueryString["action"]);
    //获得用户信息
    if (action == "get")
    {
        ShowUserInfo();
    }
    //修改用户信息
```

```
    else if (action == "update") {
        //创建 XmlDocument 对象
        XmlDocument XmlDoc = new XmlDocument();
        //加载 XML 数据流
        XmlDoc.Load(Request.InputStream);
        Response.ContentType = "text/xml";
        XmlNode xNode = XmlDoc.ChildNodes[0];
        string uid = xNode.ChildNodes[0].InnerText;
        string uname = xNode.ChildNodes[1].InnerText;
        string upwd = xNode.ChildNodes[2].InnerText;
        string uemail = xNode.ChildNodes[3].InnerText;
        string uphone = xNode.ChildNodes[4].InnerText;
        //执行更新操作
        string sql = "update users set password='" + upwd + "',email='"+uemail+"',
        phone='"+uphone+"' where id=" + uid;
        SQLHelper.ExecuteSql(sql);
        Response.Clear();
        //Session["username"] = uname;
        //调用输出用户最新资料信息的函数
        ShowUserInfo();
    }
}
//读取用户资料
private void ShowUserInfo()
{
    var uid = SQLHelper.GetUserId(Session["username"].ToString());
    string sql = "select * from users where id=" + uid;
    SqlDataReader reader = SQLHelper.ExecuteReader(sql);
    returnstr += "<users>";
    while (reader.Read())
    {
        //转换为 XML 形式
        returnstr += "<uid>" + Convert.ToString(reader["id"]) + "</uid>"
        + "<uname>" + Convert.ToString(reader["username"]) + "</uname>"
        + "<upwd>" + Convert.ToString(reader["password"]) + "</upwd>"
        + "<uemail>" + Convert.ToString(reader["email"]) + "</uemail>"
        + "<uphone>" + Convert.ToString(reader["phone"]) + "</uphone>";
    }
    reader.Close();
    reader.Dispose();
    returnstr += "</users>";
    Response.Write(returnstr);
}
```

网站后台处理用户信息的文件为 AdminUsers.aspx，该文件和 GetUserInfo.aspx 文件类似，

只是在 Page_Load()事件处理函数中，增加了一个判断语句，代码如下所示。

```
//删除指定用户记录
else if (action == "Delete")
{
    string delid = Request.QueryString["videoid"].ToString();
    string sql = "delete from Users where id=" + delid;
    SQLHelper.ExecuteSql(sql);
}
```

17.3.2　处理视频分类和视频列表程序设计

网站的前、后台都用到了视频分类，需要对视频类别进行读取，有些需要读取所有类别，有些需要读取某一条。这些操作都是通过 GetVideoClass.aspx 文件实现的，该文件的代码如下所示。

```
protected void Page_Load(object sender, EventArgs e)
       {
          string action = Convert.ToString(Request.QueryString["action"]);
          //输出所有分类
          if (action == "getclass")
          {
             string returnstr = "<classes label='分类' data='catagory'>";
             returnstr += GetClasses(0);
             returnstr += "</classes>";
             Response.Write(returnstr);
          }
          //输出上级分类
          if (action == "getParentClass")
          {
             string returnstr = "<ParentClass>";
             string sql = "select * from [ClassType] where parentid=0";
             SqlDataReader reader = SQLHelper.ExecuteReader(sql);
             while (reader.Read())
             {
                returnstr += "<mclass classname='" + Convert.ToString(reader
                ["classname"])
                   + "' classid='" + Convert.ToString(reader["id"])
                   + "' parentclassname='顶级分类'>";
                returnstr += "</mclass>";
             }
             returnstr += "</ParentClass>";
             reader.Close();
             reader.Dispose();
```

```
            Response.Write(returnstr);
        }
        //输出单个分类
        if (action == "request")
        {
            string ChangeClass = Request.QueryString["Pclass"].ToString();
            string returnstr = "<OneClass>";
            string sql = "select * from [view_classtype] where parentid="
            +GetClassID(ChangeClass);
            SqlDataReader reader = SQLHelper.ExecuteReader(sql);
            while (reader.Read())
            {
                returnstr += "<mclass classname='" + Convert.ToString(reader
                ["classname"])
                    + "' classid='" + Convert.ToString(reader["id"])
                    + "' parentclassname='" + Convert.ToString(reader["pare-
                    ntclassname"])
                    + "' parentid='" + Convert.ToString(reader["parentid"])
                    +"'>";
                returnstr += "</mclass>";
            }
            returnstr += "</OneClass>";
            reader.Close();
            reader.Dispose();
            Response.Write(returnstr);
        }
        //输出单个分类（根据上级分类 id）
        if (action == "getsonclass")
        {
            string pclassid = Request.QueryString["parentid"].ToString();
            string returnstr = "<OneClass>";
            string sql = "select * from [view_classtype] where parentid=" +
            pclassid;
            SqlDataReader reader = SQLHelper.ExecuteReader(sql);
            while (reader.Read())
            {
                returnstr += "<mclass classname='" + Convert.ToString(read-
                er["classname"])
                    + "' classid='" + Convert.ToString(reader["id"])
                    + "' parentclassname='" + Convert.ToString(reader["pare-
                    ntclassname"])
                    + "' parentid='" + Convert.ToString(reader["parentid"])
                    + "'>";
                returnstr += "</mclass>";
            }
            returnstr += "</OneClass>";
            reader.Close();
            reader.Dispose();
```

```
                Response.Write(returnstr);
            }
        }
        //递归遍历所有的分类
        string returnstr = "";
        protected string GetClasses(int TopClassID)
        {
            string sql = "select * from [ClassType] where parentid=" + TopClassID;
            SqlDataReader reader = SQLHelper.ExecuteReader(sql);
            while (reader.Read())
            {
                returnstr += "<mclass label='" + Convert.ToString(reader["clas-
                sname"])
                    + "' data='" + Convert.ToString(reader["id"])
                    + "'>";
                //递归调用该函数，遍历所有类别
                GetClasses(int.Parse(reader["id"].ToString()));
                returnstr += "</mclass>";
            }
            reader.Close();
            reader.Dispose();
            return returnstr;
        }
        //由类别名称得到该类别的编号
        protected string GetClassID(string ParentClassName) {
            string str = "";
            string sql = "select * from [classtype] where classname='"+Parent
            ClassName+"'";
            SqlDataReader reader = SQLHelper.ExecuteReader(sql);
            while (reader.Read())
            {
                str = Convert.ToString(reader["id"]);
            }
            reader.Close();
            reader.Dispose();
            return str;
        }
```

后台中，添加、修改和删除类别的处理文件为 AdminClass.aspx。该文件主要是根据获得提交的信息，判断这些信息，从而进行不同的数据库操作。这里不再显示代码。

处理视频列表和处理视频分类相似，只不过视频增加了几个字段。视频信息包括了类别编号，浏览者可以通过类别筛选视频。另外，浏览者还可以通过名称关键字搜索视频。这些需要改变查询语句。获取视频列表的查询语句代码如下所示。

```
    string sql = "select * from [view_videos_classtype]" ;
    //判断请求参数，如果是类别筛选
    if (action == "byclass")
    {
```

```
        //类别号不存在或者不是显示所有的情况
        if (classid != null && classid != "" && classid != "all")
        {
            sql += " where classid=" + classid;
        }
    }
    //请求参数为其他情况
    else {
        //关键字搜索
        if (searchstr != null && searchstr != "")
        {
            sql += " where videoname like '%" + searchstr+"%'";
        }
        //大类名称筛选
        else if (Pclass != null && Pclass != "")
        {
            sql += " where parentclassname like '%" + Pclass + "%'";
        }
    }
```

17.4 前台设计

前台是网站的“脸”，需要拿出去让人看，首先要保证界面的美感，其次就是功能，使用有趣的形式展示内容，增加用户的兴趣。本网站自定义了自己的主题，不再使用 Flex 中默认的主题，使得界面更个性化。本网站的前台中包括的特点如下所示。

- 搜索栏在网站头部时，不能移动；在主题内容区域中时，可以移动。
- 主题内容中的窗口都可以移动位置。
- 视频播放器窗口可以实现最大化、最小化、还原、关闭和缩放等效果。

下面除了讲述如何实现这些效果外，还包括了如何实现其他的基础功能模块。

17.4.1 事件处理类设计

Flex 程序中，组件与组件之间通过事件进行交互，事件分为不同的类型。在大型的项目中，要用到的事件类型比较多，实现的功能也不相同。全部使用软件自带的事件，会降低程序的可读性，有些功能可能实现起来也会比较困难。采用自定义事件类，能够解决此问题。例如 MyLoginEvent 类，该类位于 myEvents 包中，代码如下所示。

```
package myEvents
{
    import flash.events.Event;
    public class MyLoginEvent extends Event
    {
        //要求注册
        public static const WANTTOREG:String="WantToReg";
```

```
            //要求登录
            public static const WANTTOLOGIN:String="WantToLogin";
            //验证登录信息
            public static const CHECKLOGIN:String="CheckLogin";
            //登录状态发生改变
            public static const LOGINSTATECHANGE:String="LoginStateChange";
            //已经登录
            public static const LOGINED:String="Logined";
            //未登录
            public static const NOLOGINED:String="NoLogin";
            //存储验证登录时的用户名
            public var username:String;
            public function MyLoginEvent(type:String, _username:String=null)
            {
                username= username;
                super(type);
            }
        }
    }
```

上述代码中，包括 WANTTOREG、WANTTOLOGIN 等 6 个静态常量，这些常量的值为不同的字符串。外部用到 WantToReg 字符串，只需要输入 MyLoginEvent.WANTTOREG 就可以。这样做的好处是能够减少输入时的错误，也增加了可读性。例如 MyLoginEvent.WANTTOREG，很容易联想到用户注册。

username 属性，在用户登录时，存储用户名。自定义事件类最大的优势，就是存储事件触发源对象传递的参数。因为某些数据，事件接收者根本无法获得，而事件触发者可以，所以采取自定义事件类，在大型项目中显得尤为重要。

通过上述代码可以看到，自定义事件类非常简单。本网站中用到的自定义事件类文件和相关说明如表 17-6 所示。

表 17-6　自定义事件类文件和相关说明

自定义事件类文件	相关说明
AdminManageEvent.as	后台类别和视频操作类
MyChangeContainerEvent.as	搜索栏在网站头部和主题内容区域之间移动类
MyClassMenuEvent.as	前台中类别菜单单击事件类
MyCloseEvent.as	关闭播放器窗口处理事件类
MyLoadEvent.as	加载视频列表事件类
MyLoginEvent.as	用户相关事件类
MyManageEvent.as	前台用户管理菜单单击事件类
MySearchEvent.as	搜索事件类
PageChangeEvent.as	分页组件的事件类
PlayerEvent.as	播放器事件类
PlayListEvent.as	播放列表事件类
SettingEvent.as	视频调节事件类

17.4.2 用户模块设计

用户模块主要包括用户注册、登录、注销和显示登录状态等子项，其中用户注册和登录需要为用户提供输入信息的界面，注销和显示登录状态根据用户登录状态的不同，而动态改变。这 3 者分别制作成自定义组件。

1. 用户注册组件

用户注册组件的文件名为 myReg.mxml，该组件基于 TitleWindow 创建，内容为用户注册界面，效果如图 17-2 所示，布局代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" fontSize="14" wid-
th="348"
    height="328" title="用户注册" layout="absolute" creationComplete="init()"
     showCloseButton="true" close="CloseMe()">
    <!—用户注册表单-->
    <mx:Form width="100%">
        <mx:FormHeading label="注册成为本站一员"/>
        <mx:FormItem label="用户名" required="true">
            <mx:TextInput id="tiusername" x="102" y="19" maxChars="20"/>
        </mx:FormItem>
        <mx:FormItem label="密码" required="true">
            <mx:TextInput id="tipassword" x="102" y="61"
                displayAsPassword="true" maxChars="20"/>
        </mx:FormItem>
        <mx:FormItem label="重复密码" required="true">
            <mx:TextInput id="tirepassword" x="102" y="61"
                displayAsPassword="true" maxChars="20"/>
        </mx:FormItem>
        <mx:FormItem label="Email" required="true">
            <mx:TextInput id="tiemail" x="102" y="61"/>
        </mx:FormItem>
        <mx:FormItem label="验证码" required="true" width="221">
            <mx:HBox>
                <mx:TextInput x="102" y="106" width="69" id="tiidentify" key
                Down="EnterKeyCheck(event)"/>
                <mx:Label width="80" id="generate" click="generate.text=gen-
                erateCheckCode()" toolTip="看不清就点我"/>
            </mx:HBox>
        </mx:FormItem>
        <mx:FormItem>
            <mx:HBox>
                <mx:Button id="btnLogin" x="68" y="156" label="登录" click=
```

```
                    "CheckLogin()"/>
                    <mx:Button id="btnClear" x="159" y="156" label="清空" click=
                    "clearHandler()"/>
                </mx:HBox>
            </mx:FormItem>
        </mx:Form>
        <!--验证表单元素是否符合条件-->
        <mx:StringValidator source="{tiusername}" trigger="{btnLogin}"
             triggerEvent="click" requiredFieldError="用户名不能为空"
             property="text"/>
        <mx:StringValidator source="{tipassword}" trigger="{btnLogin}"
             triggerEvent="click" requiredFieldError="密码不能为空"
             property="text" />
        <mx:EmailValidator source="{tiemail}"
            triggerEvent="click"
            requiredFieldError="Email 不能为空"
            invalidCharError="Email 格式不正确"
            property="text"/>
        <!--与服务器端通信的 HTTPService 组件-->
        <mx:HTTPService id="HttpSLogin" url="Server/Reg.aspx"
             resultFormat="e4x" method="GET" result="myHandler(event)"
             showBusyCursor="true" fault="faultHandler(event)">
            <mx:request xmlns="">
                <username>{tiusername.text}</username>
                <password>{tipassword.text}</password>
                <email>{tiemail.text}</email>
            </mx:request>
        </mx:HTTPService>
    </mx:TitleWindow>
```



图 17-2　注册界面效果

注册页面中需要的自定义函数，主要包括生成验证码、初始化、单击【注册】按钮事件处理函数和 HTTPService 组件的处理函数。这里介绍一下生成验证码的函数 generateCheckCode()，其代码如下所示。

```
<mx:Script>
    <![CDATA[
        private function generateCheckCode():String{
            var num:Number;
            var code:String;
            var checkCode:String="";
            for(var i:int=0;i<5;i++){
                //产生一个 5 位数的随机整数
                num=Math.round(Math.random()*100000);
                //得到随机整数的末一位，然后与 48 的和作为 ASCII 码，求出对应字符
                code=String.fromCharCode(48+(num%10));
                checkCode +=code;
            }
            return checkCode;
        }
    ]]>
</mx:Script>
```

需要提交的页面，按回车键自动提交，能够使用户操作起来更加方便。下面讲解该功能的实现方法。首先，在组件的初始化函数中，添加键盘键按下的监听事件，代码如下所示。

```
private function init():void{
    this.addEventListener(KeyboardEvent.KEY_DOWN,EnterKeyCheck);
}
```

然后编写监听事件的处理函数 EnterKeyCheck()。该函数判断按的键的值，如果为 13（回车键对应的值），执行提交函数。代码如下所示。

```
private function EnterKeyCheck(e:KeyboardEvent):void
{
    if(e.keyCode==13){
        CheckLogin();
    }
}
```

提交成功后，HTTPService 组件的处理函数 myHandler()，根据服务器端返回的数据做出不同的操作。当返回的数据中的 state 节点值为 usernameRegisted 时，表明用户名已被注册，弹出提示窗口。当为其他情况时，从场景中移除该组件对象。代码如下所示。

```
private function myHandler(event:ResultEvent):void{
    //返回用户名已被注册
    if(event.result.state=="usernameRegisted"){
        Alert.show("该用户名已被注册","提示");
```

```
        }else{
            //注册过后，关闭注册窗口
            if(this.isPopUp){
                this.parent.removeChild(this);
            }
            Alert.show("注册成功","提示");
        }
    }
```

2. 用户登录组件

用户登录组件的文件名为 myLogin.mxml，该文件的内容和注册组件的文件相似。只不过所提交的服务器端文件不同，提交的参数不同，HTTPService 组件的处理函数也不同。

这里只讲述 HTTPService 组件的 result 事件处理函数 myHandler()。该函数得到服务器端返回的数据，判断该数据中 state 节点的值。如果为 yes，向所在应用程序发送参数为 MyLoginEvent.LOGINSTATECHANGE 的 MyLoginEvent 事件对象。如果为其他情况，就再判断应用程序中是否已经有登录信息，如果有，则向所在应用程序发送参数为 MyLoginEvent.LOGINSTATECHANGE 的 MyLoginEvent 事件对象，表明之前的登录已经过期。

```
    private function myHandler(event:ResultEvent):void{
        if(event.result.state=="yes"){
            //登录成功，向主程序发送登录状态改变事件
            this.parentApplication.dispatchEvent(new MyLoginEvent(MyLoginEvent.
            LOGINSTATECHANGE));
        }else{
            if(this.parentApplication.myusername!=""){
                //登录失败，但系统仍记录有登录信息，向主程序发送登录状态改变事件
                this.parentApplication.dispatchEvent(new MyLoginEvent(MyLoginE-
                vent.LOGINSTATECHANGE));
                }
                Alert.show("登录失败","提示");
            }
    }
```

3. 用户登录状态组件

用户登录状态组件文件名为 myLoginState.mxml，该组件实现登录前显示【新建账号】和【登录】按钮，登录后显示欢迎文字。首先添加需要的组件，代码如下所示。

```
    <mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"  creationComplete="in-
    it()">
        <!—用户登录前显示-->
        <mx:HBox id="NoLogin">
            <mx:Button label="新建账户" click="NewAcount_Click()"/>
            <mx:Button label="登录" click="Login_Click()"/>
```

```
        </mx:HBox>
        <!—用户登录后显示-->
        <mx:HBox id="Logined" visible="false">
        <mx:Label text="欢迎{this.parentApplication.myusername}"/>
        <mx:Button label="注销" click="LoginOut_Click()" />
        </mx:HBox>
        <!—向服务器端发送注销请求-->
        <mx:HTTPService id="HttpLoginOut" url="Server/LoginOut.aspx"
             resultFormat="e4x" method="GET" result="myHandler(event)"
              showBusyCursor="true" fault="faultHandler(event)">
            <mx:request xmlns="">
                <action>LoginOut</action>
            </mx:request>
        </mx:HTTPService>
    </mx:Canvas>
```

然后添加 Script 代码，主要包括初始化函数、事件监听函数和事件处理函数。初始化函数添加 2 个监听器，分别监听类型为 MyLoginEvent.LOGINED 和 MyLoginEvent.NOLOGINED 的事件，分别由 LoginedHandler()和 NoLoginedHandler()函数进行处理。这两个函数分别表示登录成功后和登录失败后的处理函数。这两个函数的代码如下所示。

```
    <mx:Script>
        <![CDATA[
            //组件初始化函数
            private function init():void{
                //监听已登录事件
                addEventListener(MyLoginEvent.LOGINED,LoginedHandler);
                //监听未登录事件
                addEventListener(MyLoginEvent.NOLOGINED,NoLoginedHandler);
            }
            //用户成功登录后处理函数
            private function LoginedHandler(event:MyLoginEvent):void{
                this.NoLogin.visible=false;
                this.Logined.visible=true;
            }
            //用户未登录处理函数
            private function NoLoginedHandler(event:MyLoginEvent):void{
                this.NoLogin.visible=true;
                this.Logined.visible=false;
            }
        ]]>
    </mx:Script>
```

17.4.3 分类模块设计

视频分类主要是方便浏览者查找，前台应该包括显示所有分类、显示某一分类的功能。这一功能无论用户登录与否都存在，做成菜单形式放在网站的头部，无疑是比较好的处理方

法。本网站采取，将显示所有分类单独做成按钮触发，显示某一分类采用下拉列表框形式。

1. 显示所有分类

在网站头部组件（Top.mxml）文件中添加一个 Button 组件，该组件的属性设置如下所示。

```
<mx:Button label="所有分类" click="ShowAllClass()"/>
```

添加单击【所有分类】按钮的事件处理函数 ShowAllClass()。该函数向自身发送类型为 MyClassMenuEvent.CHOOSECLASS 的 MyClassMenuEvent 对象，该对象的 Classid 属性值为 all。代码如下所示。

```
<mx:Script>
    <![CDATA[
    private function ShowAllClass():void{
        this.dispatchEvent(new MyClassMenuEvent(MyClassMenuEvent.CHOOSECLA-
        SS,"all"));
    }
    ]]>
</mx:Script>
```

由于该事件需要和播放列表进行交互，所以该事件处理函数没有编写在该组件中，而是在主程序中。

2. 显示某一分类

在能够实现显示某一分类功能之前，需要从数据库中读取所有类别，这在头部组件的初始化函数中进行。变量的初始化和组件初始化函数的代码如下所示。

```
<mx:Script>
    <![CDATA[
        //存储类别菜单数据源
        [Bindable]
        private var menuBarCollection:XMLListCollection;
        private var menubarXML:XMLList;
        private function initCollections():void {
            this.HttpGetVideoClass.send();
            //省略部分代码
        }
    ]]>
</mx:Script>
```

在头部组件文件中添加一个 MenuBar 组件，用来显示所有类别。该组件的属性设置如下所示。

```
<mx:MenuBar dataProvider="{menuBarCollection}" labelField="@label"
itemClick="ClickMenu_Class(event)">
```

添加单击菜单项的事件处理函数 ClickMenu_Class()。该函数向自身发送类型为

MyClassMenuEvent.CHOOSECLASS 的 MyClassMenuEvent 对象，该对象的 Classid 属性值为所单击菜单项数据源的 data 节点。代码如下所示。

```
private function ClickMenu_Class(event:MenuEvent):void{
    this.dispatchEvent(new MyClassMenuEvent(MyClassMenuEvent.CHOOSECLASS,
    event.item.@data));
}
```

17.4.4 搜索模块设计

搜索模块实现的功能主要包括关键字搜索、可游离和在主题内容区域可拖动。根据这些功能，需要将网站头部和主题内容区域中的搜索栏分别做成组件。这样实现起来就比较简单。

1. 公用组件

将头部搜索栏和主题内容区域相区别，是为了实现可游离。不过这两者之间实现的搜索功能是相同的，都是进行关键字搜索，所以将搜索关键字功能单独组成组件，供两者调用。公用组件的文件名为，代码如下所示。

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" verticalAlign="middle">
    <mx:Script>
        <![CDATA[
            import myEvents.*;
            private function doSearch():void{
                this.parentApplication.dispatchEvent(new MySearchEvent(MyS-
                earchEvent.SEARCH,tiSearchText.text));
            }
        ]]>
    </mx:Script>
    <mx:Label text="搜索"/>
    <mx:TextInput id="tiSearchText"/>
    <mx:Button click="doSearch()" useHandCursor="true" buttonMode="true" sty-
    leName="btsearch"/>
</mx:HBox>
```

2. 网站头部中的搜索栏

此搜索栏组件比较简单，主要能够实现单击搜索框左面的按钮，搜索栏移到主题内容区域中的功能。代码如下所示。

```
<mx:HBox  xmlns:mx="http://www.adobe.com/2006/mxml"  verticalAlign="middle"
horizontalGap="3" xmlns:ns1="myComponents.*">
    <mx:Script>
        <![CDATA[
            import myEvents.*;
```

```
                private function ShouShuoClick():void{
                    //向应用程序发送 MyChangeContainerEvent 事件对象
                    this.parentApplication.dispatchEvent(new MyChangeContainer
                    Event(MyChangeContainerEvent.CHANGETOSTAGE));
                }
            ]]>
        </mx:Script>
        <mx:Button click="ShouShuoClick()" styleName="downArrow" buttonMode="tr-
        ue" useHandCursor="true"/>
        <ns1:mySearchBarContent>
        </ns1:mySearchBarContent>
    </mx:HBox>
```

3. 主题内容区域中的搜索栏

此搜索栏除了可以移到网站头部外，还可以拖住搜索按钮右边的区域，拖动整个搜索栏，效果如图 17-3 所示。

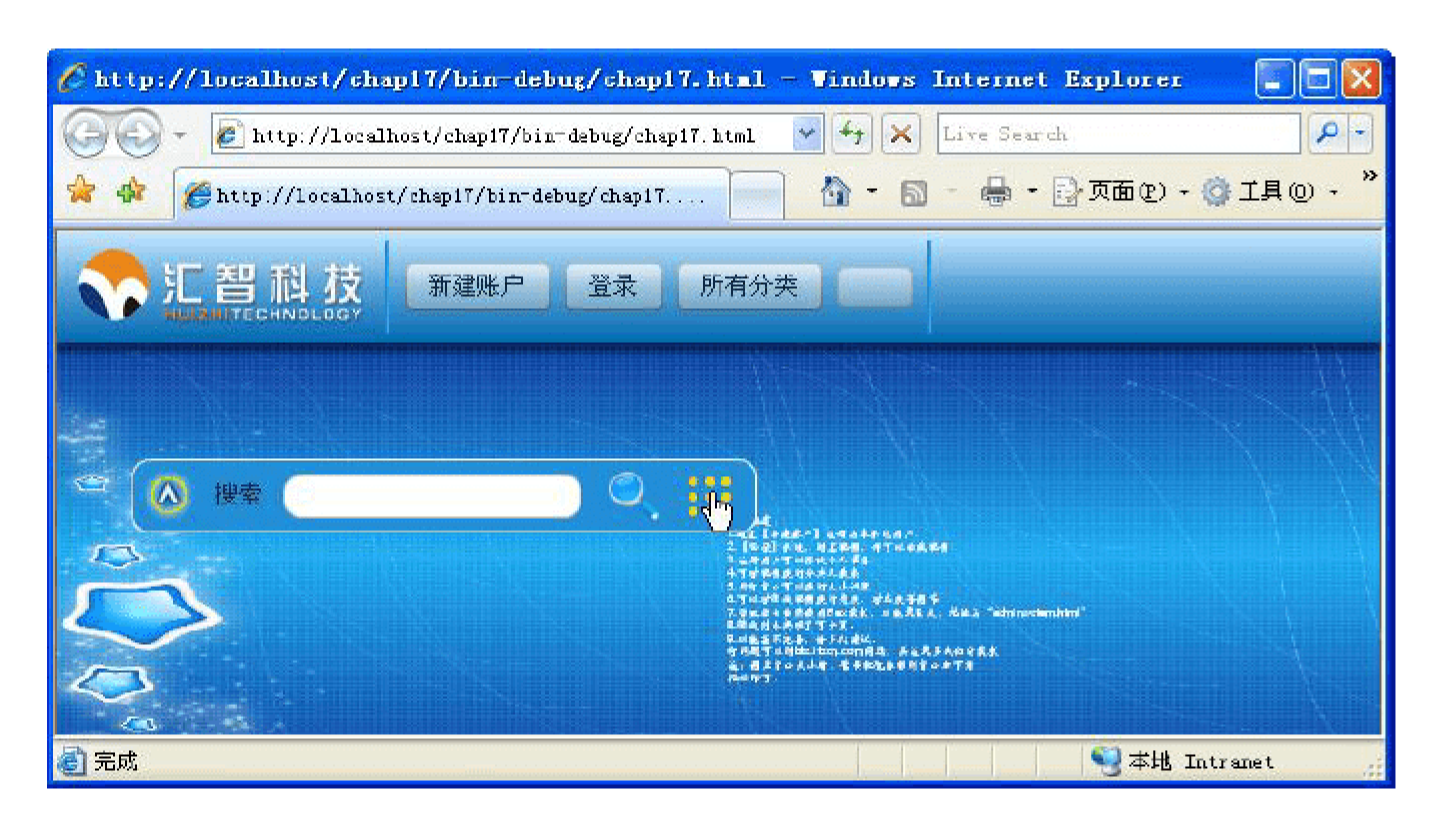


图 17-3 主题内容区域中的搜索栏效果

实现拖动的拖动代码如下所示。

```
    <mx:Script>
            <![CDATA[
                import myEvents.*;
                // 拖动时所需变量
                private var dragStartMouseX:Number;     //开始鼠标的 x
                private var dragStartMouseY:Number;     //开始鼠标的 y
                private var dragStartX:Number;          //开始拖动时的坐标 x
                private var dragStartY:Number;          //开始拖动时的坐标 y
                private var dragMaxX:Number;            //可拖动最大的坐标 x
                private var dragMaxY:Number;            //可拖动最大的坐标 y
                private function init():void{
                    this.DragToMove.addEventListener(MouseEvent.MOUSE_DOWN, on
                    MouseDownTitleBar);
```

```
            }
            //当鼠标按住 titlebar 不放时, 可以拖动
            private function onMouseDownTitleBar(event:MouseEvent):void{
                //开始拖动, 初始化 x,y
                //panel 所在 x,y
                dragStartX = x;
                dragStartY = y;
                //鼠标所在 x,y
                dragStartMouseX = parent.mouseX;
                dragStartMouseY = parent.mouseY;
                //拖动的距离
                dragMaxX = parent.width - width;
                dragMaxY = parent.height - height;
                //开始拖动的时候, 添加以下两个事件
                stage.addEventListener(MouseEvent.MOUSE MOVE, onMouseMove);
                stage.addEventListener(MouseEvent.MOUSE UP, onMouseUp);
            }
            //拖动时触发的方法
            private function onMouseMove(event:MouseEvent):void{
                //确保不脱出屏幕
                var targetX:Number = Math.min(dragMaxX, dragStartX + (parent.
                mouseXdragStartMouseX));
                x = Math.max(0, targetX);
                var targetY:Number = Math.min(dragMaxY, dragStartY + (parent.
                mouseYdragStartMouseY));
                y = Math.max(0, targetY);
            }
            //拖动后松开鼠标
            private function onMouseUp(event:MouseEvent):void{
                //拖动结束移除事件
                stage.removeEventListener(MouseEvent.MOUSE_MOVE, onMouseMo-
                ve);
                stage.removeEventListener(MouseEvent.MOUSE_UP, onMouseUp);
            }
            //移到头部按钮的单击事件处理函数
            private function ShouShuoClick():void{
                //记录移走前的坐标
                this.parentApplication.movesearchbarx=this.x;
                this.parentApplication.movesearchbary=this.y;
                //向主程序发送 CHANGETOTOP 类型的事件
                this.parentApplication.dispatchEvent(new MyChangeContainer-
                Event(MyChangeContainerEvent.CHANGETOTOP));
            }
        ]]>
    </mx:Script>
```

17.4.5　视频列表模块设计

视频列表窗口使用 TitleWindow 组件，可以拖动和改变大小，效果如图 17-4 所示。视频列表显示使用 TileList 组件，采用图文形式（上方缩略图，下方名称）。鼠标移到图片上方，显示播放按钮。视频列表窗口在改变大小时，视频列表会根据宽度自动换行，防止出现横向滚动条。下面一个一个功能地进行实现。首先添加 TileList、myPager 和 HTTPService 组件，代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns="myComponents.*" xmlns:mx="http://www.adobe.com/2006/
mxml"
     horizontalScrollPolicy="off" creationComplete="init()"
     showCloseButton="true" close="CloseVideoList()">
    <mx:TileList itemRenderer="myComponents.myVideoListItemRender" id="tlVi-
    deoList" verticalAlign="top" width="100%" styleName="playlist" vertica-
    lScrollPolicy="off"horizontalScrollPolicy="off"/>
    <myPager id="pagebar" PageChange="changepage(event)" bottom="0">
    </myPager>
    <mx:HTTPService id="HttpGetVideoList" url="Server/GetVideoList.aspx"
        resultFormat="e4x" method="GET" result="myHandler(event)"
        showBusyCursor="true" fault="faultHandler(event)">
        <mx:request xmlns="">
            <classid>{nowclassid}</classid>
            <searchstr>{searchtext}</searchstr>
            <action>{myaction}</action>
        </mx:request>
    </mx:HTTPService>
</mx:TitleWindow>
```



图 17-4　视频列表效果

上述代码中用到的 myPager 组件，为自定义的分页组件。使用方法是，指定其 PageChange 事件处理函数，该函数为分页组件提供数据源。数据源需要在视频列表组件创建完成时获得，数据源分为通过分类筛选和搜索结果两种。视频列表组件创建完成时的处理函数，主要是添加事件监听器。代码如下所示。

```
<mx:Script>
    <![CDATA[
        private function init():void{
            //加载列表信息完成监听器
            addEventListener(MyLoadEvent.LOADCOMPLETE,HandleLoadComplete);
            //关键字搜索事件监听器
            addEventListener(MySearchEvent.SEARCH,SearchHandler);
            //类别筛选事件监听器
            addEventListener(MyClassMenuEvent.CHOOSECLASS,ChooseClassHand-
            ler);
            //鼠标按下事件监听器，主要实现改变大小
            addEventListener(MouseEvent.MOUSE_DOWN, resizeHandler);
            //鼠标按下标题栏事件监听器，主要实现拖动和突出显示
            titleBar.addEventListener(MouseEvent.MOUSE_DOWN, titlebar-
            MousedownHandler);
            dispatchEvent(new MyLoadEvent(MyLoadEvent.LOADCOMPLETE,par-
            entApplication.xmlNodeList));
        }
    ]]>
</mx:Script>
```

上述代码中添加的监听器类型和实现的功能主要是在 MyLoadEvent.LOADCOMPLETE 类型实现加载列表数据完成时，显示列表内容，在 MySearchEvent.SEARCH 类型实现关键字搜索时，重新加载列表数据，在 MyClassMenuEvent.CHOOSECLASS 类型实现分类筛选时，重新加载列表数据。

MouseEvent.MOUSE_DOWN 类型实现缩放窗口时，除了改变大小外，还能够使得视频列表会根据宽度自动换行。在 titleBar 组件上添加的 MouseEvent.MOUSE_DOWN 类型的监听器，实现拖动标题栏移动窗口，以及在主场景中突出显示功能。事件监听器的处理函数具体如下所示。

```
private function HandleLoadComplete(event:MyLoadEvent):void{
    //获得加载的 XML 数据
    var myxmllist:XMLList=event.LoadedXML;
    //将 XML 数据转换为数据形式
    var arr:Array=new Array();
    for each(var obj:Object in myxmllist){
        arr.push(obj);
    }
    XMLVideoListArray=arr;
    //设置分页组件，实现数据分页
```

```
        pagebar.RecordCount=XMLVideoListArray.length;
        pagebar.Open();
        //初始化视频列表的列
        if(pagebar.RecordCount>4){
            tlVideoList.columnCount=4;
        }else{
            tlVideoList.columnCount=pagebar.RecordCount;
        }
        //初始化视频列表的行
        if(pagebar.RecordCount>pagebar.mPageSize){
            if(pagebar.mPageSize%tlVideoList.columnCount==0){
                tlVideoList.rowCount=pagebar.mPageSize/tlVideoList.
                columnCount;
            }else{
                tlVideoList.rowCount=pagebar.mPageSize/tlVideoList.columnCount
                +1;
            }
        }else{
            tlVideoList.rowCount=pagebar.RecordCount/4+1;
        }
        //初始化视频列表的宽度和高度，防止出现横向滚动条
        tlVideoList.width=tlVideoList.columnCount*110;
        tlVideoList.height=tlVideoList.rowCount*140;
        //初始化视频列表的位置，在主场景中居中
        this.x=this.parentApplication.width/2-this.width/2;
        this.y=this.parentApplication.height/2-this.height/2-60;
    }
    //改变大小处理函数
    public function resizeHandler(event:MouseEvent):void
    {
        //获得窗口的右下角坐标
        var lowerLeftX:Number = x + width;
        var lowerLeftY:Number = y + height;
        //以窗口右下角向上 7×7 后的坐标
        var upperLeftX:Number = lowerLeftX-7;
        var upperLeftY:Number = lowerLeftY-7;
        //获得鼠标基于主场景的位置
        var panelRelX:Number = event.localX + x;
        var panelRelY:Number = event.localY + y;
        //判断鼠标是否在右下角 7×7 区域内
        if (upperLeftX <= panelRelX && panelRelX <= lowerLeftX)
        {
            if (upperLeftY <= panelRelY && panelRelY <= lowerLeftY)
            {
                event.stopPropagation();
```

```
                var rbEvent:MouseEvent = new MouseEvent(RESIZE_CLICK,true);
                //传递主场景的坐标到 MouseEvent 事件
                rbEvent.localX = event.stageX;
                rbEvent.localY = event.stageY;
                dispatchEvent(rbEvent);
            }
        }
    }
    //搜索处理函数
    private function SearchHandler(event:MySearchEvent):void{
        searchtext=event.SearchText;
        this.myaction="bykeyword";
        this.HttpGetVideoList.send();
    }
    //搜索结果处理函数
    private function SearchLoadCompleteHandler(event:Event):void{
        if(this.visible==false){
            this.visible=true;
        }
        var xmlNodeList:XMLList=new XMLList(event.target.data).video;
        var arr:Array=new Array();
        for each(var obj:Object in xmlNodeList){
            if(obj.name.indexOf(searchtext)>-1){
                arr.push(obj);
            }
        }
        XMLVideoListArray=arr;
        var newpagechangeevent:PageChangeEvent=new PageChangeEvent(0,pagebar.mPa-
        geSize);
        tlVideoList.dataProvider=newpagechangeevent.Filter(XMLVideoListArray);
        pagebar.RecordCount=XMLVideoListArray.length;
        pagebar.Open();
    }
```

myPage 组件的使用，需要指定 PageChange 事件处理函数。本视频列表中指定的函数为 changepage()，代码如下所示。

```
    private function changepage(event:PageChangeEvent):void{
        tlVideoList.dataProvider=event.Filter(XMLVideoListArray);
    }
```

视频播放列表的项，是在 TileList 组件中的 itemRenderer 属性中指定的，是名称为 myVideoListItemRender 的自定义组件。该组件的代码如下所示。

```
    <mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" paddingBottom="5" padd-
    ingLeft="5"
        paddingRight="5" paddingTop="5" xmlns:ns1="myComponents.*"
```

```
        height="140" width="110" horizontalScrollPolicy="off" verticalScrollPo-
        licy="off">
        <ns1:VideoListPicBox id="picbox" srcobj="{data}" width="100" height=
        "100">
        </ns1:VideoListPicBox>
        <mx:Label text="{data.@videoname}" x="10" y="108"/>
    </mx:VBox>
```

在视频缩略图区域中，当鼠标移到上方时，显示播放按钮，移开时隐藏播放按钮。因此在此组件中创作，不是太容易。所以本网站中，单独做成组件，名称为 VideoListPicBox，代码如下所示。

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="100" height=
"100">
    <mx:Script>
        <![CDATA[
            import myEvents.PlayListEvent;
            [Bindable]
            public var srcobj:Object;
            //单击播放按钮，向主程序发送 CHANGEPLAY 类型事件
            private function toPlayme():void{
this.parentApplication.dispatchEvent(new PlayListEvent(PlayListEvent.CHANG-
EPLAY,srcobj));
            }
        ]]>
    </mx:Script>
    <mx:Image source="{srcobj.@picsrc}"    width="100" height="100"
        mouseOver="canvasplay.visible=true;"/>
    <mx:Canvas width="100" height="100" visible="false" id="canvasplay"
    mouseOut="canvasplay.visible=false;">
        <mx:Button horizontalCenter="0" verticalCenter="0" click="toPlayme()"
        styleName="btplaybig" buttonMode="true" useHandCursor="true"/>
    </mx:Canvas>
</mx:Canvas>
```

17.4.6　收藏夹及个人信息模块设计

收藏夹及个人信息在网站中是固定的内容，前者从数据库中读取登录用户的收藏视频，后者读取用户个人信息。本网站中将这两者归到用户中心，放在页面的头部靠右位置，采用下拉列表框的形式呈现。网站头部组件中，用户中心的代码如下所示。

```
<mx:MenuBar dataProvider="{menuManageXML}" id="menuManage"labelField="@label"
itemClick="ClickMenu_User(event)" />
```

该下拉列表框的数据从 UserMenuList.xml 文件中读取，该文件的内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
    <menuitem label="用户中心" data="catagory">
```

```
        <menuitem label="管理收藏" data="Favorite"/>
        <menuitem label="个人信息" data="Person"/>
    </menuitem>
```

单击用户中心菜单项时，触发 ClickMenu_User() 函数，该函数向主程序发送 MyManageEvent 事件对象。事件对象的类型为 MyManageEvent.CLICKMENU，Classid 属性的值为菜单项对应 XML 数据的 data 属性。代码如下所示。


```
private function ClickMenu_User(event:MenuEvent):void{
    this.parentApplication.dispatchEvent(new MyManageEvent(MyManageEvent.C-
    LICKMENU,event.item.@data));
}
```

17.4.7 整合主程序

程序中用到的组件创建完成后，需要将这些组件安排到主程序中，并实现其通信。本网站页面分为头部和主题内容区域，头部名称为 Top 的自定义组件，该组件基于 Application ControlBar 创建。将头部单独作为组件，主要是为了减少主程序中的代码量。

Top 组件主要包括了网站的 logo，用户注册和登录按钮，类别菜单，搜索框及用户中心等组件，效果如图 17-5 和图 17-6 所示。许多组件已在上文中介绍过，这里不再详细讲解 Top 组件的内容。



图 17-5　未登录的头部效果

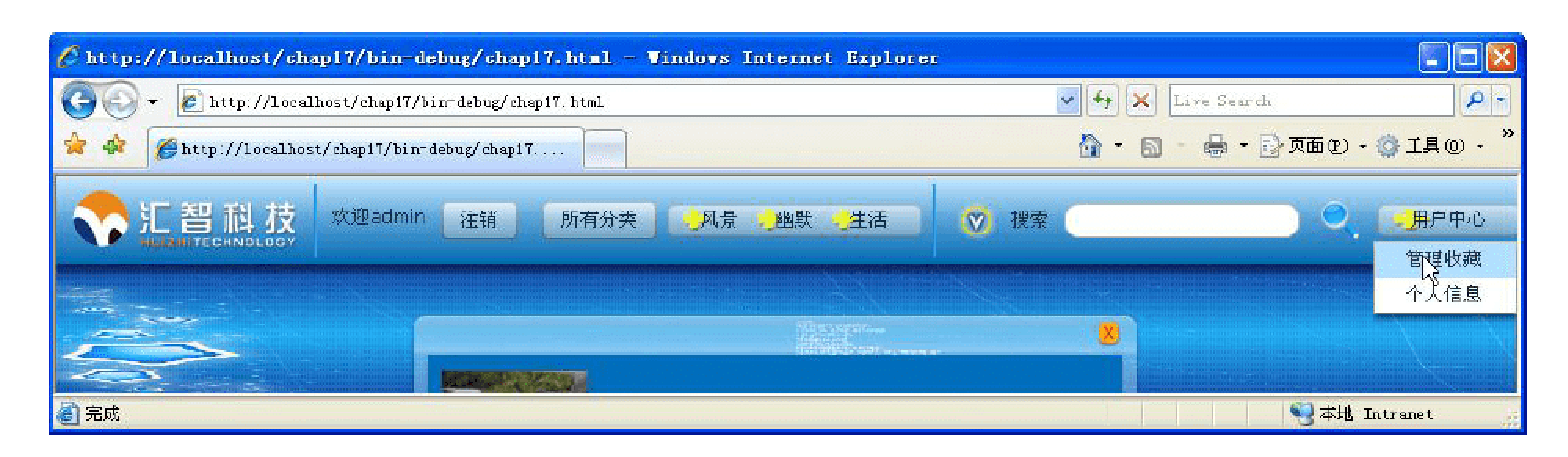


图 17-6　登录后的头部效果

网站的主题内容区域是一个 Canvas 组件，该组件的属性设置及内容如下所示。

```
<mx:Canvas  x="0"  y="0"  width="100%"  height="100%"  id="canvasMainContent"
```

```
    horizontalScrollPolicy="off" verticalScrollPolicy="off">
        <ns1:CheckLoginState x="74" y="0" id="myCheckLoginState">
        </ns1:CheckLoginState>
    </mx:Canvas>
```

上述代码中可以看到，该组件只包括 CheckLoginState 组件，实现的功能是在初始化时确定用户的登录状态。

主程序中组件间通信，需要在主程序创建完成后通过添加事件监听器实现。主程序中，创建完成后的处理函数代码如下所示。

```
protected function creationCompleteHandler():void{
    //为主程序添加类型为 RESIZE_CLICK 的事件监听器，主要实现窗口的大小调节
    addEventListener(RESIZE_CLICK, resizeHandler);
    //播放选定的视频监听器
    addEventListener(PlayListEvent.CHANGEPLAY, ChangePlayHandler);
    //搜索栏要求由头部到主题内容区域
    addEventListener(MyChangeContainerEvent.CHANGETOSTAGE, ChangeToStageHa-
    ndler);
    //搜索栏要求由主题内容区域到头部
    addEventListener(MyChangeContainerEvent.CHANGETOTOP, ChangeToTopHandler);
    //关键字搜索事件监听器
    addEventListener(MySearchEvent.SEARCH,SearchHandler);
    //要求登录监听器
    addEventListener(MyLoginEvent.WANTTOLOGIN,WanttologinHandler);
    //要求注册监听器
    addEventListener(MyLoginEvent.WANTTOREG,WanttoregHandler);
    //登录状态发生改变监听器
    addEventListener(MyLoginEvent.LOGINSTATECHANGE,LoginStateChangeHandler);
    //登录成功监听器
    addEventListener(MyLoginEvent.LOGINED,LoginedHandler);
    //未登录监听器
    addEventListener(MyLoginEvent.NOLOGINED,NoLoginedHandler);
    //关闭播放器窗口监听器
    addEventListener(PlayerEvent.CLOSEWIN,ClosewinHandler);
    //用户中心菜单选择监听器
    addEventListener(MyManageEvent.CLICKMENU,UserManageMenuHandler);
    //初始化组件
    this.canvasMainContent.addChildAt(videllist_main,this.canvasMainConten
    t.numChildren-1);
    this.mail_top.addEventListener(MyClassMenuEvent.CHOOSECLASS,ChooseCla-
    ssHandler);
    HttpGetVideoList.send();
    this.movesearchbarx=this.canvasMainContent.width/2-100;
    this.movesearchbary=this.canvasMainContent.height/2-15;
    this.myCheckLoginState.HttpCheckLoginState.send();
```

```
    }
```

从上述代码中，可以看到添加的事件监听器和初始化组件语句。其中，初始化组件语句在主题内容区域添加播放列表，该播放列表显示所有的视频。初始化主题内容区域的搜索框位置，居于区域中间。

主程序中用户登录和注册的处理函数，主要包括登录状态改变、已经登录、未登录、要求登录和要求注册等，具体代码如下所示。

```
    private function LoginStateChangeHandler(event:MyLoginEvent):void{
        this.myCheckLoginState.HttpCheckLoginState.send();
    }
    private function LoginedHandler(event:MyLoginEvent):void{
        myusername=event.username;
        this.mail_top.dispatchEvent(new MyLoginEvent(MyLoginEvent.LOGINED));
    }
    private function NoLoginedHandler(event:MyLoginEvent):void{
        myusername="";
        this.mail_top.dispatchEvent(new MyLoginEvent(MyLoginEvent.NOLOGINED));
    }
    private function WanttologinHandler(event:MyLoginEvent):void{
        var newmyLoginwin:IFlexDisplayObject=PopUpManager.createPopUp(this,myL-
        ogin,true);
        PopUpManager.centerPopUp(newmyLoginwin);
    }
    private function WanttoregHandler(event:MyLoginEvent):void{
        var newmyRegwin:IFlexDisplayObject=PopUpManager.createPopUp(this,myReg,
        true);
        PopUpManager.centerPopUp(newmyRegwin);
    }
```

对用户中心菜单的处理，主要是根据所单击菜单项进行的。如果是收藏夹，在主题内容区域显示收藏夹窗口，效果如图 17-7 所示，代码如下所示。

```
    private function UserManageMenuHandler(event:MyManageEvent):void{
        switch(event.mItem){
            //收藏夹
            case "Favorite":
                try{
                    this.canvasMainContent.removeChild(this.myuserfavoritewin);
                }catch(e:Error){

                }
                myuserfavoritewin=new UserFavorite();
    this.canvasMainContent.addChildAt(this.myuserfavoritewin,this.canvasMainC-
    ontent.numChildren-1);
                break;
```

```
            //个人信息
            case "Person":
                try{
                    this.canvasMainContent.removeChild(this.myuserinfowin);
                }catch(e:Error){

                }
                myuserinfowin=new UserInfo();
    this.canvasMainContent.addChildAt(this.myuserinfowin,this.canvasMainContent.
    numChildren-1);
                break;
        }
    }
```

单击视频列表中的某一播放按钮，需要播放对应的视频，效果如图 17-8 所示。单击收藏夹中的某一浏览按钮，实现同样功能。这在主程序中通过监听 PlayListEvent.CHANGEPLAY 类型事件实现。该监听事件的处理函数代码如下所示。

```
    private function ChangePlayHandler(e:PlayListEvent):void{
        //移除之前的播放器
        try{
            this.canvasMainContent.removeChild(myplayer);
        }catch(er:Error){

        }
        //创建并添加新的浏览器组件，并初始化该浏览器组件的属性
        myplayer=new Player();
        this.canvasMainContent.addChildAt(myplayer,this.canvasMainContent.num-
        Children-1);
        this.canvasMainContent.setChildIndex(myplayer,this.canvasMainContent.n
        umChildren-1);
        myplayer.x=this.canvasMainContent.width/2-200;
        myplayer.y=this.canvasMainContent.height/2-100;
        myplayer.nowPlayingobj=e.video;
        myplayer.myPlayerContent.sliderVideoPosition.value=0;
        try{
            myplayer.myPlayerContent.myVideoDisplay.playheadTime=0;
        }catch(ex:Error){

        }
        myplayer.myPlayerContent.InitPlayer(true);
    }
```

搜索框在页面头部和主题内容区域间游离，是通过主程序中添加的 MyChangeContainer Event 事件监听器实现的，该事件的处理函数如下所示。

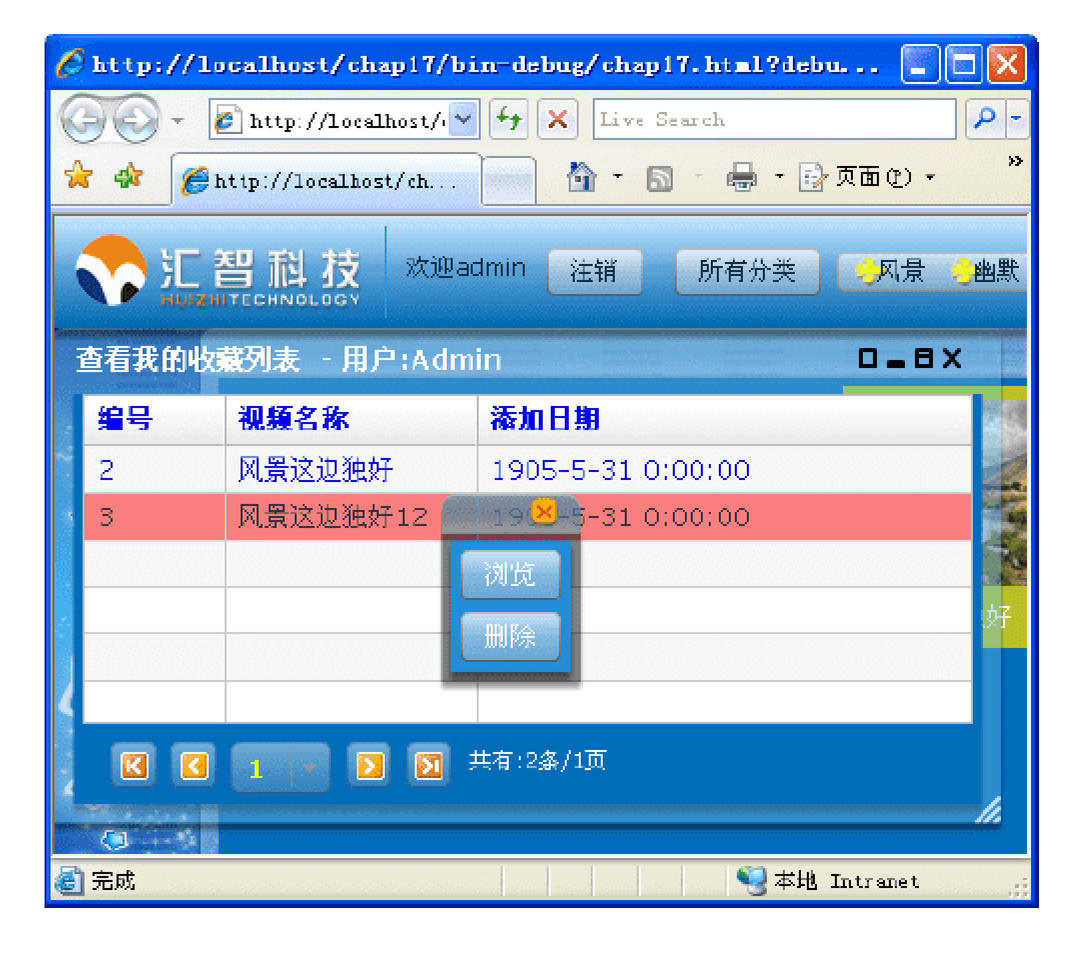



图 17-7　弹出的收藏夹效果

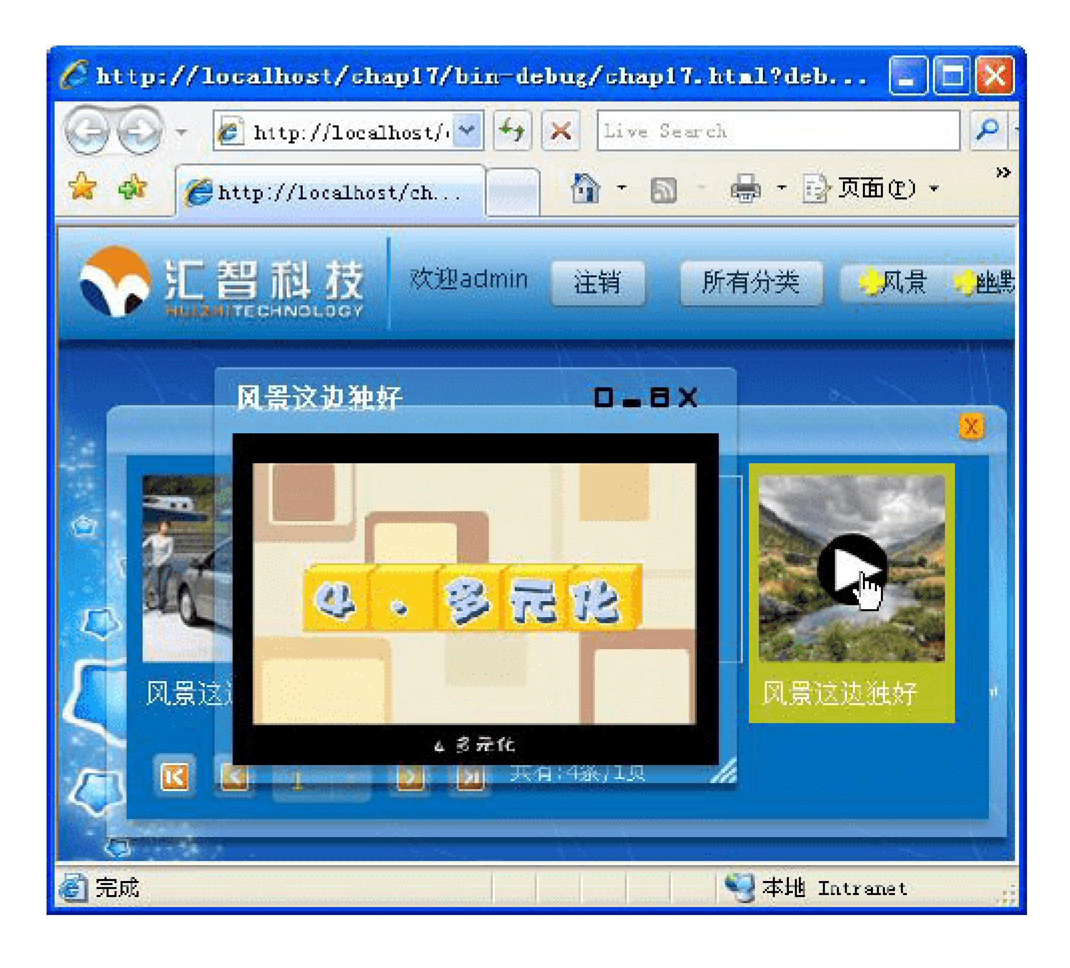


图 17-8　单击播放按钮，播放视频

```
//移动到主题内容区域
private function ChangeToStageHandler(e:MyChangeContainerEvent):void{
    this.mail_top.removeChild(this.mail_top.mySearchBar_1);
    this.canvasMainContent.addChild(this.mymovesearchbar);
this.canvasMainContent.setChildIndex(this.mymovesearchbar,this.canvasMainC
ontent.numChildren-1);
    mymovesearchbar.x=movesearchbarx;
    mymovesearchbar.y=movesearchbary;
}
//移动到页面头部
private function ChangeToTopHandler(e:MyChangeContainerEvent):void{
    this.canvasMainContent.removeChild(this.mymovesearchbar);
    this.mail_top.addChild(this.mail_top.mySearchBar_1);
}
```

浏览者通过搜索或分类筛选，得到视频列表。这是通过在主程序中添加 MySearchEvent 和 MyClassMenuEvent 事件监听器实现的。当浏览者搜索或单击分类菜单时，主程序监听到这些事件，并通知视频列表组件，更新列表信息。监听事件的处理函数如下所示。

```
private function SearchHandler(e:MySearchEvent):void{
    //在主题内容区域的最高层，添加视频列表
    this.canvasMainContent.addChildAt(this.videllist_main,this.canvasMain
    Content.numChildren-1);
    //向视频列表发送 MySearchEvent 事件对象
    this.videllist_main.dispatchEvent(new MySearchEvent(MySearchEvent.SEAR-
    CH,e.SearchText));
}
private function ChooseClassHandler(e:MyClassMenuEvent):void{
    this.canvasMainContent.addChildAt(this.videllist_main,this.canvasMain-
    Content.numChildren-1);
```

```
        //向视频列表发送 MyClassMenuEvent 事件对象
        this.videllist_main.dispatchEvent(new MyClassMenuEvent(MyClassMenuEvent.
        CHOOSECLASS,e.Classid));
    }
```


17.5　后台设计

后台作为管理员管理网站的平台，功能主要包括用户管理、类别管理和视频管理等。用户管理主要对前台注册用户进行审核、编辑和删除，比较简单，不再讲解。类别管理主要包括添加、修改和删除类别，视频管理和类别管理类似。

后台页面的文件名为 AdminSystem.mxml，所需要的功能模块组件文件和说明如表 17-7 所示。后台的难点在于修改类别、添加视频和修改视频 3 个功能模块。

表 17-7　后台页面中功能模块组件文件和说明

文件名	说明
AdminAddNewClass.mxml	添加类别组件
AdminAddNewVideo.mxml	添加视频组件
AdminAllParentClass.mxml	修改类别和视频时的大类下拉列表框组件
AdminChangButton.mxml	修改和删除类别时的按钮组件
AdminClassManage.mxml	修改和删除类别组件
AdminParentClass.mxml	添加类别时的大类下拉列表框组件
AdminSonClass.mxml	修改视频时的子类下拉列表框组件
AdminVideoChangButton.mxml	修改和删除视频时的按钮组件
AdminVideoManage.mxml	修改和删除视频组件

17.5.1　添加类别

添加类别比较简单，通过 Form 和 HTTPService 组件很容易实现。页面比较简单，只有 1 个文本框、1 个下拉列表框和两个按钮，效果如图 17-9 所示。代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" showCloseButton=
"true" close="CloseMe()" xmlns="myComponents.*"  title="添加分类">
    <mx:Form>
        <mx:FormItem label="分类名称: ">
            <mx:TextInput id="txtID"/>
        </mx:FormItem>
        <mx:FormItem label="上级分类: ">
            <AdminParentClass id="comboxparentclass"/>
        </mx:FormItem>
        <mx:FormItem>
```

```
            <mx:HBox>
                <mx:Button label="确定" click="AddNewClassClick()"/>
                <mx:Button  label="关闭" click="CloseMe()"/>
            </mx:HBox>
            </mx:FormItem>
        </mx:Form>
        <mx:HTTPService id="httpSaveNewClass" url="Server/AdminClass.aspx"
             resultFormat="e4x" method="GET" result="myHandler(event)"
             showBusyCursor="true" fault="faultHandler(event)">
            <mx:request>
                <action>Insert</action>
                <classname>{txtID.text}</classname>
                <pclassid>{comboxparentclass.selectedid}</pclassid>
            </mx:request>
        </mx:HTTPService>
    </mx:TitleWindow>
```



图 17-9 【添加分类】窗口效果

当单击【确定】按钮后，HTTPService 组件向服务器端发送请求，保存添加的类别信息。这些代码比较简单，不再讲解。上级分类的选择，需要用到 AdminParentClass 组件。AdminParentClass 组件中声明了两个主要变量 ParentAry 和 selectedid。前者用来储存所有上级分类信息，作为下拉列表框的数据源。后者储存选择项对应的类别编号（id），当向数据库中保存时，读取的上级类别编号就是该值。AdminParentClass 组件的主要内容如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <mx:ComboBox xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="I-
    nitAllClass()"
        dataProvider="{ParentAry}" labelField="@classname" change="parentclass-
        change()">
    <mx:Script>
        <![CDATA[
            [Bindable]
```

```
            private var ParentAry:Array;
            [Bindable]
            public var selectedid:String;
            //创建完成时，从服务器端获得所有上级分类信息
            private function initAllClass():void
            {
                HttpGetVideoClass.send();
            }
            //对从服务器端获得的信息进行处理，并初始化 selectedid 变量
            private function myHandler(event:ResultEvent):void{
                var xmlClassList:XMLList=new XMLList(event.result).mclass;
                var arr:Array=new Array();
                for each(var obj:Object in xmlClassList){
                    arr.push(XML(obj));
                }
                ParentAry=arr;
                selectedid=ParentAry[0].@classid;
            }
            //当下拉列表框的选择项改变时，改变 selectedid 的值
            private function parentclasschange():void{
                this.selectedid=ParentAry[this.selectedIndex].@classid;
            }
        ]]>
    </mx:Script>
        <mx:HTTPService id="HttpGetVideoClass" url="Server/GetVideoClass.aspx"
             resultFormat="e4x" method="GET" result="myHandler(event)"
             showBusyCursor="true" fault="faultHandler(event)">
            <mx:request>
                <action>getParentClass</action>
                <num>{rndNum}</num>
            </mx:request>
        </mx:HTTPService>
    </mx:ComboBox>
```

17.5.2　修改和删除类别

修改和删除类别放到一个组件中，该组件中使用 DataGrid 组件显示类别信息，包括类别编号、所属分类以及修改后保存和删除操作，效果如图 17-10 所示。该 DataGrid 组件的设置代码如下所示。

```
    <mx:DataGrid  id="dgclasslist" width="100%" rowHeight="25" editable="true"
    doubleClickEnabled="true">
        <mx:columns>
            <mx:DataGridColumn headerText="编号" dataField="@classid" width="90"
```

```
            editable="false"/>
            <mx:DataGridColumn headerText="所属分类" dataField="@parentclassname"
            itemEditor="myComponents.AdminAllParentClass" />
            <mx:DataGridColumn headerText="分类名称" dataField="@classname"/>
            <mx:DataGridColumn headerText="操作" editable="false" itemRenderer=
            "myComponents.AdminChangButton" />
        </mx:columns>
    </mx:DataGrid>
```

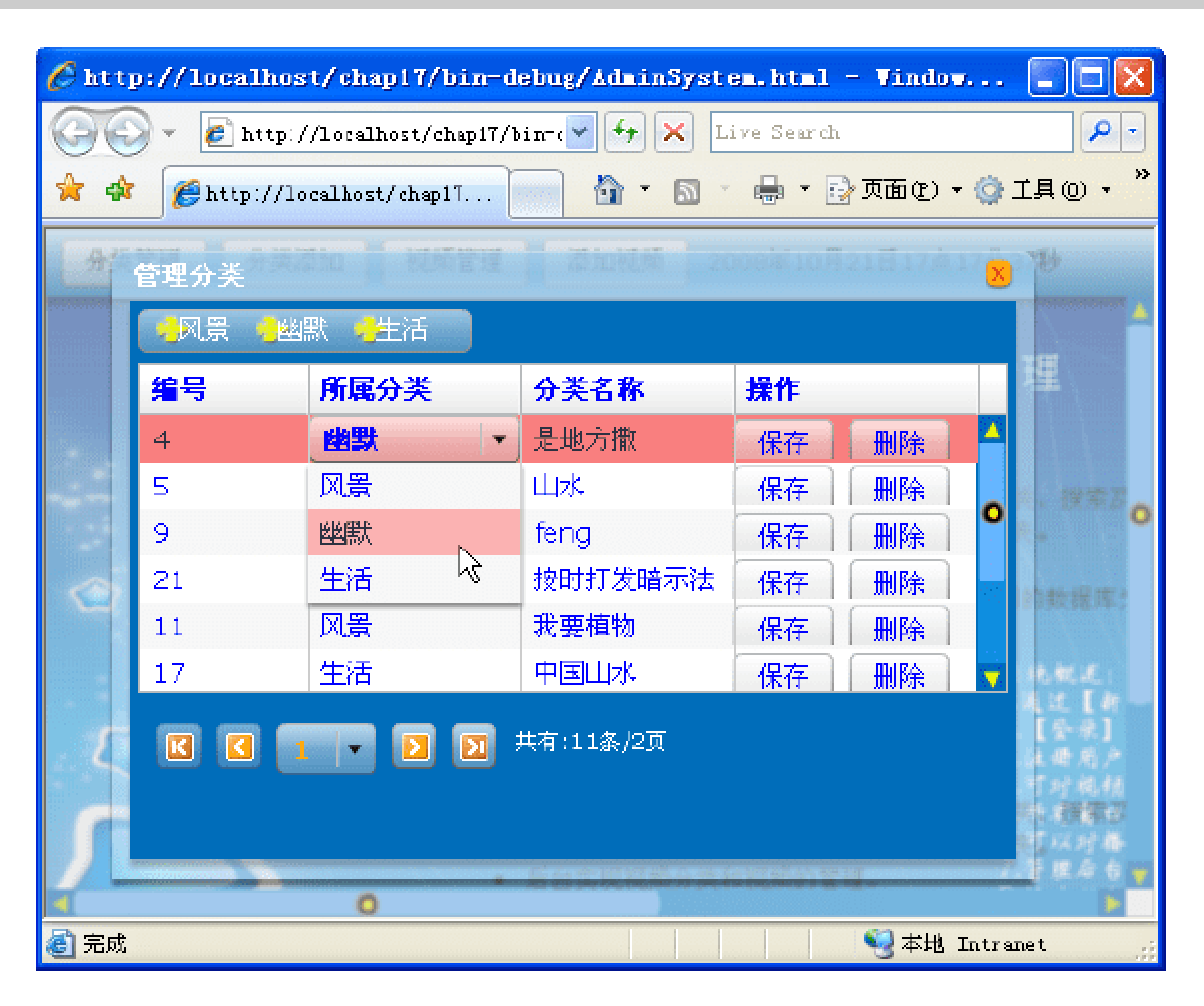


图 17-10　修改和删除类别窗口效果

所属分类列中，itemEditor 属性的值为 AdminAllParentClass 组件，该组件和上文中讲到的 AdminParentClass 组件相似。但有一点不同，此处所属分类选择项改变时，需要改变该条记录上级类别编号。AdminAllParentClass 组件的 change 事件处理函数的代码如下所示。

```
    private function parentclasschange():void{
        data.@parentid=this.ParentAry[this.selectedIndex].@classid;
    }
```

17.5.3　添加视频

添加视频较为复杂，涉及到父类和子类的二级联动，以及图片和视频的上传操作。界面中元素较多，主要是一个 Form 和一个 HTTPService。Form 中包括类别下拉列表框、文本框和按钮等，生成后的效果如图 17-11 所示，代码如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
    title="添加视频"
    showCloseButton="true" close="CloseMe()" creationComplete="init()">
    <mx:XML id="VideoObj">
         <video>
```

```
        <classid>{cmbclasstype.selectedItem.@classid}</classid>
        <videoname>{tivideoname.text}</videoname>
        <picsrc>flv/{tipicsrc.text}</picsrc>
        <videosrc>flv/{tivideosrc.text}</videosrc>
        </video>
    </mx:XML>
       <mx:Form>
           <mx:FormItem label="视频名称" required="true">
               <mx:TextInput id="tivideoname"  />
           </mx:FormItem>
           <mx:FormItem label="视频大类" required="true">
               <mx:ComboBox id="cmbparentclasstype" change="parentClassChange
               ()"
                   dataProvider="{parentclasstypearr}" labelField="@classname">
               </mx:ComboBox>
           </mx:FormItem>
           <mx:FormItem label="视频子类" required="true">
               <mx:ComboBox id="cmbclasstype"
                   dataProvider="{classtypearr}" labelField="@classname">
               </mx:ComboBox>
           </mx:FormItem>
           <mx:FormItem label="视频图片" required="true">
               <mx:HBox width="100%">
                   <mx:TextInput id="tipicsrc" editable="false" />
                   <mx:Button label="浏览" click="BrowsePicFiles()"/>
                   <mx:Button id="btnUploadpic" label="上传" click="UploadPic()"
                   enabled="false"/>
               </mx:HBox>
           </mx:FormItem>
           <mx:FormItem label="视频地址" required="true">
               <mx:HBox width="100%">
                   <mx:TextInput id="tivideosrc" editable="false"  />
                   <mx:Button label="浏览" click="BrowseVideoFiles()"/>
                   <mx:Button id="btnUpload" label="上传" click="UploadVideo()"
                   enabled="false"/>
               </mx:HBox>
           </mx:FormItem>
           <mx:FormItem >
               <mx:HBox width="100%">
                   <mx:Button label="确定" click="SubmitVideo()"/>
                   <mx:Button label="取消"  click="CloseMe()"/>
               </mx:HBox>
           </mx:FormItem>
           <mx:ProgressBar id="progressbar" labelPlacement="center" trackHeight=
           "15" height="20"  color="#000000" width="330"/>
```

```
        </mx:Form>
        <mx:HTTPService id="HttpgetParentClass" url="Server/GetVideoClass.aspx"
             resultFormat="e4x" method="GET" result="myHandler(event)"
             showBusyCursor="false" fault="faultHandler(event)">
            <mx:request xmlns="">
                <action>{myClassAction}</action>
                <parentid>{parentclassid}</parentid>
            </mx:request>
        </mx:HTTPService>
    </mx:TitleWindow>
```

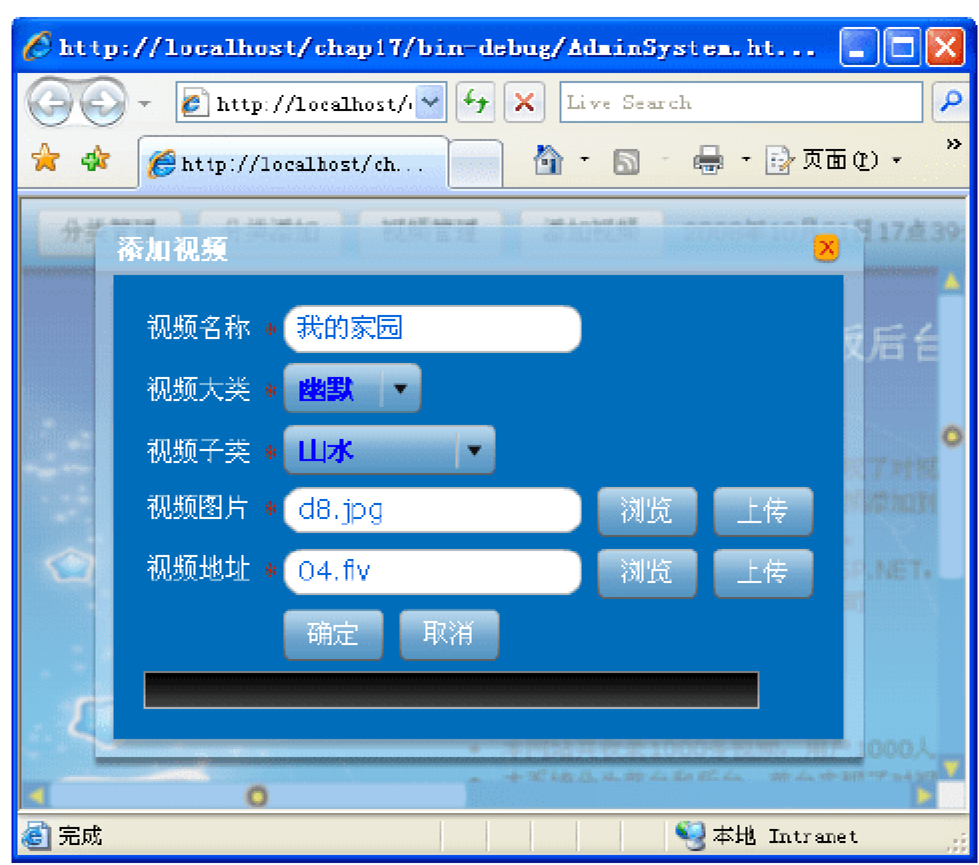


图 17-11　添加视频窗口效果

父类和子类的二级联动实际上也是组件与组件的交互，本实例主要是通过绑定变量的形式实现的。主要实现代码如下所示。

```
    <mx:Script>
        <![CDATA[
            //声明父类数据源
            [Bindable]
            private var parentclasstypearr:XMLList;
            //声明子类数据源
            [Bindable]
            private var classtypearr:XMLList;
            //声明 HTTPService 组件所要发送的 action 参数值
            [Bindable]
            private var myClassAction:String;
            //组件创建时从服务器端获得父类信息
            private function init():void{
                myClassAction="getParentClass";
                HttpgetParentClass.send();
            }
            //将从服务器端获得的类别信息，根据 myClassAction 的值，赋予不同变量
```

```
            private function myHandler(event:ResultEvent):void{
                if(myClassAction=="getParentClass"){
                    parentclasstypearr=new XMLList(event.result).mclass;
                    parentclassid=parentclasstypearr[0].@classid;
                    myClassAction="getsonclass";
                    HttpgetParentClass.clearResult(true);
                    HttpgetParentClass.send();
                }else{
                    classtypearr=new XMLList(event.result).mclass;
                }
            }
            //父类选择项发生改变时，更新子类信息
            private function parentClassChange():void{
    this.parentclassid=this.parentclasstypearr[this.cmbparentclasstype.select-
    edIndex].@classid;
                myClassAction="getsonclass";
                this.HttpgetParentClass.send();
            }
        ]]>
    </mx:Script>
```

17.5.4　修改和删除视频

修改和删除视频与修改和删除类别相似，同样放到一个组件中，该组件中使用 DataGrid 显示类别信息，包括视频编号、所属大类、所属子类以及修改后保存和删除操作，效果如图 17-12 所示。

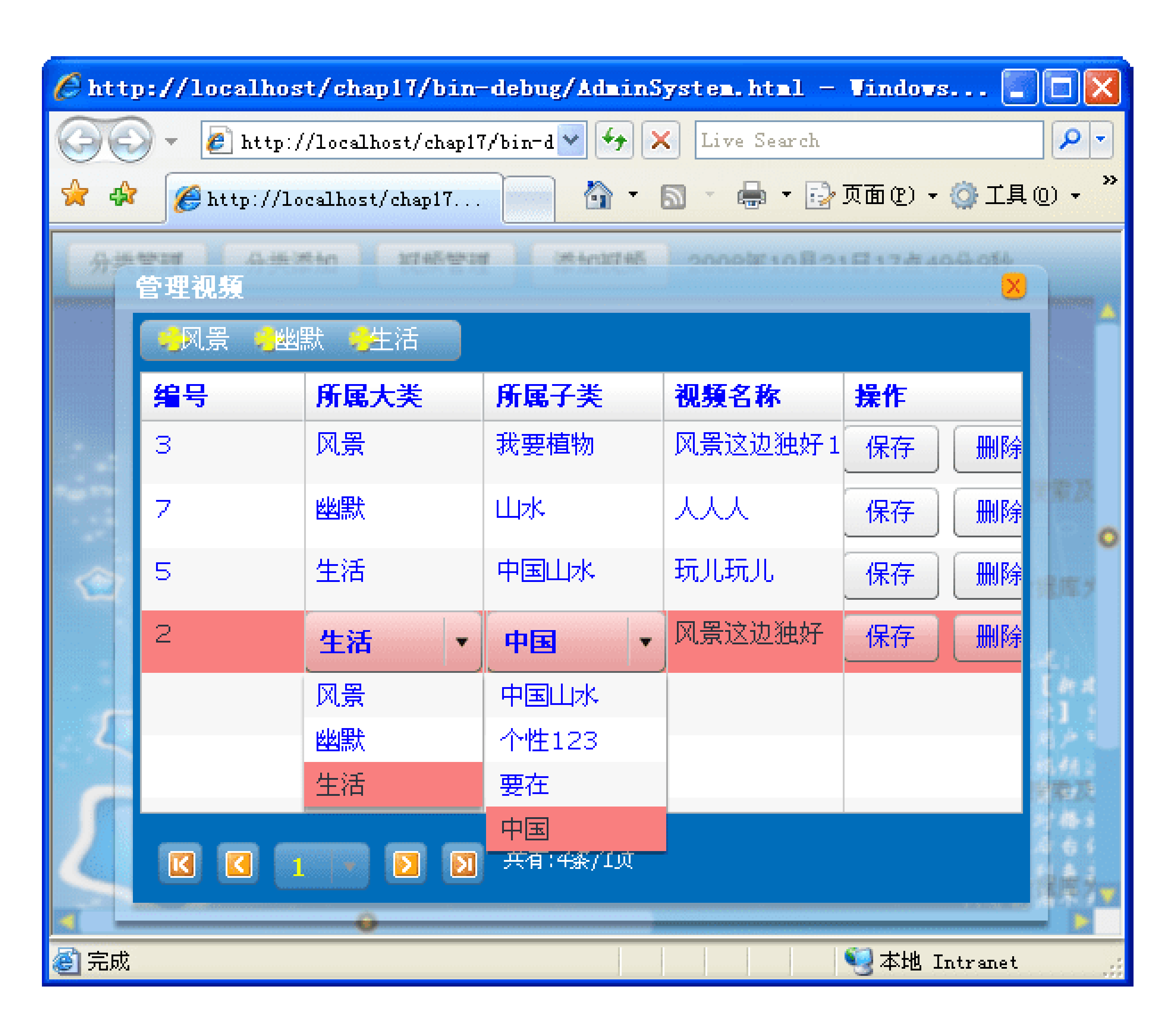


图 17-12　修改和删除视频窗口效果

17.5.5 整合主程序

主程序主要包括了头部和主题内容区域的代码。头部包括 4 个按钮，单击后弹出不同的窗口。主题内容区域显示系统概述等信息，效果如图 17-13 所示。头部的代码如下所示。

```
<mx:ApplicationControlBar width="100%"  dock="true">
    <mx:Button label="分类管理" click="PopupClass()"/>
    <mx:Button label="分类添加" click="AddNewClass()"/>
    <mx:Button label="视频管理" click="VideoManage()"/>
    <mx:Button label="添加视频" click="PopupAddVideo()"/>
    <mx:Label text="{nowtime}"/>
    <mx:Spacer width="280" height="20"/>
</mx:ApplicationControlBar>
```

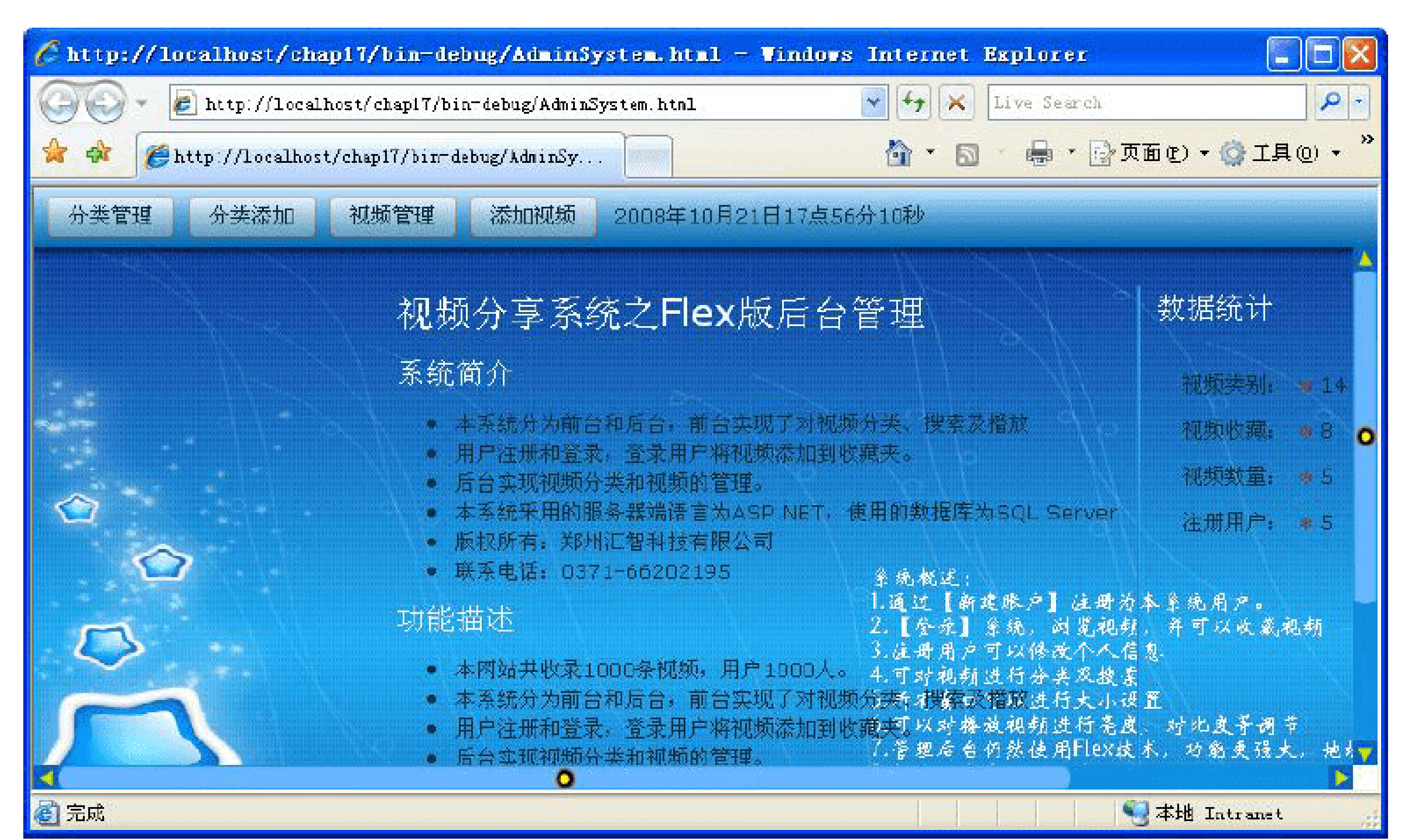


图 17-13　后台主程序界面效果

主程序创建完成时，执行 initApp()函数。该函数从服务器端获得网站统计信息，包括视频类别、视频收藏、视频、注册用户数量。添加 3 个事件监听器，分别实现修改类别和视频后更新列表以及更新头部时间的功能。